XML: UNO STANDARD IN ESPANSIONE

Ernesto Damiani
Paolo Fezzi

XML, inizialmente proposto come standard per la rappresentazione dei documenti testuali, si sta ora imponendo come infrastruttura generale del World Wide Web, per l'interscambio di informazioni tra le diverse applicazioni. XML stabilisce un insieme di convenzioni per definire diversi linguaggi di codifica, in base alle diverse tipologie di informazioni da rappresentare: associando ai dati di partenza informazioni descrittive (dette "tag" o marcatori), pone le premesse per la costruzione di inferenze automatiche sulla semantica dei dati (Semantic Web).

1. UNO STANDARD PER LA RAPPRESENTAZIONE DEI TESTI

XML (*eXtensible Markup Language*) è uno standard abbastanza recente, nato per la rappresentazione di documenti testuali: ideato da Tim Bray e Jean Pauli, è stato pubblicato ufficialmente nel febbraio 1998 dal *World Wide Web Council* (W3C), il consorzio preposto alla definizione degli standard del World Wide Web.

1.1. Struttura logica e presentazione

Le componenti elementari di un testo, i caratteri, sono, com'è noto, rappresentate nel computer mediante il codice ASCII (*American Standard Code for Information Interchange*). Per rappresentare la qualità grafica con cui i caratteri sono stampati su carta o visualizzati sullo schermo (per precisare cioè il tipo e le dimensioni dei font, caratteristiche come neretto, corsivo ecc.) sono nati specifici formati di videoscrittura (per esempio Word), formati di stampa (PostScript) e di visualizzazione (PDF, *Portable Document Format*) e veri e propri linguaggi per la stampa, come TROFF e Tex (molto usato nelle università per la stampa di formule matematiche).

XML si distingue rispetto a questi formati di visualizzazione e di stampa, non solo in quanto *standard documentato e indipendente dai produttori di software*, ma soprattutto perché risponde ad un'esigenza specifica, avvertita da editori e studiosi umanisti, quella cioè *di rappresentare la struttura logica di un documento* (per esempio la suddivisione in titoli, capitoli, paragrafi ecc.), *creando un'astrazione della sua presentazione*, ovvero dell'aspetto grafico, con cui il documento si presenta sulla pagina stampata e sullo schermo di un computer. Al contrario di TROFF e Tex, *linguaggi procedurali*, che descrivono le operazioni da fare per stampare le diverse porzioni di un documento, XML è un *linguaggio dichiarativo*, che definisce le varie parti del testo e le loro rispettive relazioni.

Più precisamente *XML è un linguaggio di markup* mediante il quale, è possibile associare a ciascuna parte di un testo, un marca-

tore (tag), che la qualifica come un determinato elemento logico (per esempio titolo, paragrafo, nota, citazione ecc., oppure, nella figura 2, nome, cognome, indirizzo ecc.), senza preoccuparsi di come esso apparirà fisicamente nel documento.
Ciascun marcatore XML è delimitato da due simboli di inizio e di fine, le parentesi angolari <> (come in <nome> e <cognome> della figura 2) e può avere degli attributi, con determinati valori (nella figura 2 è il caso del tag <loc> il cui attributo tipo ha il valore "via").
Un *elemento testuale* è generalmente delimitato da un tag di apertura e uno di chiusura[1]: quest'ultimo, omonimo rispetto a quello di apertura, si distingue per il simbolo /, posposto alla parentesi angolare < (nell'esempio del curriculum vitae di figura 2, la stringa "Bianchi" può essere ad esempio identificata come elemento cognome, con la codifica XML <cognome>Bianchi</cognome>).
Un documento XML ha una *struttura gerarchica*: ciascun elemento, delimitato da un tag di apertura e di chiusura, è contenuto in altri elementi, ed un elemento radice (nell'esempio in figura 2 l'elemento <curriculum> </curriculum>) contiene tutti gli altri.
I marcatori XML ricordano da vicino le glosse con le quali i copisti medievali commentavano e integravano i manoscritti: una sorta di "*etichette*" *informatiche*, utili per classificare le porzioni di testo alle quali si riferiscono e assegnare loro un ben preciso significato, che può diventare oggetto di elaborazioni da parte del computer.
Un documento XML è un file di testo, basato sulla codifica ASCII a 7 bit (cioè sui 128 caratteri di base). L'estensione di questa codifica di base, per esigenze di internazionalizzazione (codifica di documenti in lingue neolatine con vocali accentate, in greco, cirillico, cinese, giapponese ecc.), può essere gestita, secondo lo standard Unicode a 32 bit (di cui l'ASCII è un sottoinsieme), con l'inserimento nel testo dei codici Unicode decimali o esadecimali dei caratteri, delimitati da un carattere

Curriculum Vitae di Carlo Bianchi

Dati personali

Nome	Carlo
Cognome	Bianchi
Residenza	via Moscova 40 - 20121 Milano (MI)
e-mail	carlo.bianchi@mail.it
tel.	02-3456789
cell.	349-286652
Nato il	24-5-1965 a Milano (MI)
Nazionalità	Italiana
Stato civile	Coniugato

Esperienze professionali

1990-1993	Info Expert s.r.l - Rho
Ruolo:	Analista Programmatore
Attività:	Sviluppo di Sistemi Esperti per la pianificazione della produzione industriale

FIGURA 1
Un esempio di documento testuale, in formato ASCII: un curriculum vitae

```
<?xml version="1.0" encoding="ISO-8859-1"
standalone="no"?>
<!DOCTYPE curriculum SYSTEM "curriculum.dtd">
<curriculum>
  <dati_personali>
    <nome>Carlo</nome>
    <cognome>Bianchi</cognome>
    <indirizzo>
      <loc tipo="via">Moscova</loc>
      <civico>40</civico>
      <cap>20121</cap>
      <comune>Milano</comune>
      <provincia>MI</provincia>
    </indirizzo>
    <email>carlo.bianchi@mail.it</email>
    <tel>02-3456789</tel>
    <cellulare>349-286652</cellulare>
    <fax/>
    <nascita>
      <data>24-5-1965</data>
      <luogo>
        <comune>Milano</comune>
        <provincia>MI</provincia>
      </luogo>
    </nascita>
    <nazionalita>Italiana</nazionalita>
    <stato_civile>Coniugato</stato_civile>
  </dati_personali>
</curriculum>
```

FIGURA 2
Codifica XML del curriculum vitae

di inizio (&) e fine (;), ad es. con à o à per l'a accentata (à), oppure con la dichiarazione del *set di caratteri* usato (nella figura 2 l'uso del set di caratteri dell'alfabeto latino di base è dichiarato nella prima riga di intestazione <?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>).
La modalità di rappresentazione grafica di ciascun elemento logico del testo, associato

[1] XML ammette per altro anche elementi vuoti, privi di testo, in cui il marcatore di apertura e di chiusura possono essere legalmente sostituiti da un unico tag, con la sintassi </tag>.

ad una coppia di marcatori, è definita a parte da appositi linguaggi, detti *fogli di stile*: diventa così possibile utilizzare uno stesso documento codificato in XML per *diverse modalità di pubblicazione* (carta, CD-ROM, World Wide Web, audio), semplicemente cambiando il foglio di stile associato. Il linguaggio di stile più usato e più sofisticato è XSL (*eXtensible Stylesheet Language*) (Paragrafo 6), un vero e proprio linguaggio di programmazione basato su regole di trasformazione, che consentono non solo di decidere il formato grafico, ma anche di scegliere quali elementi visualizzare e in che ordine. Sono utilizzati anche i più semplici fogli di stile CSS (*Cascading Style Sheet*), che si limitano ad associare a ciascun elemento XML determinate caratteristiche di formattazione (tipo e dimensione dei font ecc.).

1.2. Tipologie di documenti e DTD

XML è un *metalinguaggio* generico che consente di costruire diversi linguaggi di codifica, ciascuno per una diversa tipologia di documenti: XML non prescrive i nomi dei diversi marcatori, ma solo la sintassi generica per la loro definizione e il loro utilizzo nell'identificazione degli elementi di testo.

In XML è possibile definire la peculiare struttura logica che identifica e descrive una tipologia di documenti nella cosiddetta DTD (*Document Type Definition*)[2], un insieme di specifiche che, situate in una sezione all'inizio del documento codificato o in un documento a parte, stabiliscono quali sono i nomi ammissibili per i marcatori, quali i nomi dei loro attributi e quali relazioni di inclusione possono sussistere tra di loro. Ad esempio, una DTD può stabilire che una determinata tipologia di documenti possa contenere al suo interno diversi elementi logici, denominati capitoli, che hanno al loro interno paragrafi e sottoparagrafi e che ciascun elemento sia identificato da un attributo "numero", con il relativo valore numerico.

In una DTD (Figura 3) la dichiarazione <!ELEMENT associa il nome di un elemento ai suoi possibili contenuti. L'elemento radice coincide con il nome associato alla DTD nella dichiarazione <!DOCTYPE che, all'interno del documento, richiama la corrispondente DTD (Figura 2). Nell'esempio del curriculum vitae l'elemento radice, curriculum, che identifica la DTD, contiene quattro elementi, dati_personali, istruzione, esperienze_professionali, interessi, che a loro volta contengono ulteriori elementi, secondo una struttura gerarchica ad albero (Figura 3).

Un documento che rispetta la sintassi generica di XML, senza riferirsi a una specifica DTD, è detto *ben formato*. Un documento XML congruente rispetto a una determinata DTD è detto *valido*. Il controllo della validità di un documento rispetto a una DTD precedentemente definita è una garanzia di congruenza formale, necessaria per applicare con successo al documento stesso procedure automatiche di elaborazione (fogli di stile per la formattazione, istruzioni di information retrieval ecc.).

```
<!ELEMENT curriculum
(#PCDATA|dati_personali|istruzione|espe
rienze_professionali|interessi)*>
<!ELEMENT dati_personali
(#PCDATA|nome|cognome|indirizzo|email|tel|cellulare|fax
|nascita|nazionalita|stato_civile)*>
<!ELEMENT istruzione
(#PCDATA|corso|formazione_professionale|lingue)*>
<!ELEMENT corso
(#PCDATA|anno|titolo|voto|azienda|org|periodo)*>
<!ELEMENT org
(#PCDATA|ragione_sociale|comune)*>
<!ELEMENT ragione_sociale
(#PCDATA)>
<!ELEMENT formazione_professionale
(#PCDATA|corso)*>
<!ELEMENT periodo
(#PCDATA|anno|da)*>
<!ELEMENT lingue (#PCDATA|lingua)*>
<!ELEMENT lingua (#PCDATA)>
<!ELEMENT esperienze_professionali
(#PCDATA|esperienza)*>
<!ELEMENT esperienza
(#PCDATA|periodo|org|ruolo|attivita|competenze)*>
<!ELEMENT ruolo (#PCDATA)>
<!ELEMENT attivita (#PCDATA)>
<!ELEMENT competenze (#PCDATA)>
<!ELEMENT interessi (#PCDATA|p)*>
<!ELEMENT emph (#PCDATA)>
```

FIGURA 3
DTD del curriculum vitae

1.3. Cenni storici: da SGML a XML, via HTML

Dall'evoluzione di GML (*Generalized Markup Language*), ideato nel 1969 dai ricercatori Goldfarb, Mosher e Lorris nei laboratori del-

[2] Una metodologia alternativa alla DTD per definire la struttura di una tipologia di documenti è costituita da XML Schema (Paragrafo 3).

ratori dell'IBM, nel 1980 nasce SGML (*Standard Generalized Markup Language*) su specifiche dell'ANSI (*American National Standards Institute*), l'ente deputato negli USA alla definizione degli standard: adottato nel 1983 dal DoD (Department of Defense), il Dipartimento della Difesa statunitense, e approvato nel 1986 dall'ISO, come standard ISO 8879.

SGML può essere considerato come il diretto progenitore di XML, con il quale ha in comune la maggior parte delle convenzioni sintattiche e semantiche per la costruzione dei tag e la possibilità di definire diversi linguaggi di codifica in base alla tipologia di documento, cioè di definire diverse DTD. La differenza consiste nella maggiore complessità sintattica di SGML e in una certa macchinosità dei suoi fogli di stile.

Alla fine degli anni '80, le maggiori organizzazioni internazionali degli informatici umanisti, ACH (*Association for Computers and the Humanities*), ACL (*Association for Computational Linguistics*) e ALLC (*Association for Literary and Linguistic Computing*), hanno deciso di utilizzare SGML per un ambizioso progetto di codifica dei testi umanistici, denominato TEI (*Text Encoding Initiative*), che è approdato alla definizione di una omonima DTD, specificamente dedicata all'elaborazione dei testi letterari, storici e filosofici: recentemente ne è stata pubblicata una versione più semplice e ridotta, denominata *TEI Lite*.

Usato da alcune case editrici americane, da enti governativi (soprattutto negli USA) e da grandi aziende per gestire ampie basi di documenti elettronici, per la sua complessità SGML non è stato, tuttavia, adottato al di fuori delle grandi organizzazioni e della ristretta cerchia dei ricercatori umanisti.

L'applicazione più popolare di SGML, più propriamente, la sua più diffusa DTD, è HTML (*HyperText Markup Language*), appositamente pensato per pubblicare documenti ipertestuali in Internet. Ideato nel 1989 e pubblicato nel 1991 da un ricercatore del CERN (*Conseil European pour la Recherché Nucleaire*) di Ginevra, Tim Berners-Lee, per la sua semplicità, HTML è in pochi anni diventato il formato standard per la pubblicazione di documenti in Internet, costituendo la base del World Wide Web, la "ragnatela" ipertestuale che ha reso Internet popolare, alla portata di tutti gli utenti.

HTML aveva, in origine, la finalità di rappresentare la struttura logica di documenti ipertestuali, ma la tumultuosa crescita del Web e le esigenze pratiche di centinaia di migliaia di redattori di pagine Web lo hanno di fatto trasformato in uno strumento di pubblicazione rapida dei documenti, in cui è andata perduta la distinzione tra struttura logica e aspetto grafico del testo, con una mescolanza inestricabile di marcatori logici (nella figura 4 <H1>, che identifica un titolo di primo livello) e grafici (nella figura 4 <FONT>, che stabilisce tipo, dimensioni e colore dei caratteri). Non essendo però sufficientemente in grado di rappresentare la struttura testuale, l'esigenza di specificare il complesso logico di differenti tipologie di documenti, in modo distinto rispetto alla loro presentazione sul Web, ha portato alla nascita di XML.

XML si può in effetti considerare una versione semplificata di SGML, definita in base all'esperienza intercorsa nel frattempo di HTML, tenendo cioè conto delle esigenze di pubblicazione e visualizzazione rapida delle pagine Web: può, ad esempio fare a meno, eventualmente, della definizione di una DTD (XML ben formato). In tal modo, nella gestione dei documenti XML, è possibile separare nettamente la loro visualizzazione (*browsing*) dall'analisi sintattica della loro congruenza rispetto a una DTD (parsing): di conseguenza, diventa più agevole lo sviluppo, a

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
3.2 Final//EN">
<HTML>
<HEAD>
    <TITLE>Curriculum Vitae di Carlo Bianchi</TITLE>
</HEAD>
<BODY>
<H1>Curriculum Vitae di Carlo Bianchi</H1>
        <FONT FACE="Arial" SIZE="+2"
COLOR="Red">Dati personali</FONT>
<TABLE>
<TR>
<TD>Nome</TD><TD>Carlo</TD>
</TR>
<TR>
<TD>Cognome</TD><TD><B>Bianchi</B></TD>
</TR>
```

FIGURA 4
Il curriculum vitae in HTML

basso costo, di strumenti software per la sola visualizzazione (*browser*).
La maggiore diffusione di XML[3] rispetto al suo progenitore, SGML, è dunque conseguenza di un contesto mutato dalla presenza di HTML, che da un lato ha reso popolari in tutto il mondo, tramite il Web, le fondamentali convenzioni dei linguaggi di markup, dall'altro costituisce uno dei possibili output di XML, grazie ai fogli di stile XSLT (*eXtensible Stylesheet Language Transformation*) (Paragrafo 6).

1.4. XML e teoria del testo

In generale, non tutte le componenti di un fenomeno sono rappresentate nel suo modello, che è costruito mediante un procedimento di astrazione, selezionando gli elementi considerati rilevanti per l'indagine: come ogni altro modello, la codifica XML di un testo ha, dunque, un valore euristico ed è relativa al punto di vista adottato dall'osservatore e ai suoi obiettivi. Di conseguenza, sono possibili molte diverse codifiche XML per uno stesso testo, mai identiche al testo stesso, cioè mai in grado di esaurirne tutti gli aspetti.
Per porre nei termini corretti la questione metodologica dell'adeguatezza e pertinenza della codifica XML rispetto al testo rappresentato, giova accennare qui in breve alla fondamentale distinzione di Hjelmslev tra *espressione* (il "significante", ossia l'aspetto visibile del testo) e *contenuto* (il "significato", cioè la semantica del testo).
Non è questa la sede per una trattazione sistematica delle possibili implicazioni per il *markup* XML delle teorie strutturaliste del testo e del linguaggio (De Saussure, Hjelmslev, Jakobson, Lotman, Propp, Greimas ecc.): ci si limita, in questo contesto, ad osservare che una cooperazione interdisciplinare tra l'informatica e le discipline umanistiche, che studiano i testi, sarebbe notevolmente feconda per lo sviluppo di XML. Potrebbe infatti contribuire da un lato a una più profonda comprensione dello statuto teorico del *markup*; dall'altro potrebbe introdurre una maggiore consapevolezza nella prassi corrente della codifica XML di documenti testuali, che spesso oscilla (confondendole in una stessa DTD) tra codifica dell'espressione e del contenuto. Nell'esempio del curriculum vitae (Figure 2 e 3) coesistono volutamente, a scopo didattico, elementi di tipo semantico, che codificano ciò di cui parla il testo (per esempio <nome>, <cognome>, <ruolo> ecc.) ed elementi che riguardano invece la forma visibile del testo (<emph>, che contrassegna le parti evidenziate del testo), ma una mescolanza del genere andrebbe evitata.

2. EVOLUZIONE DI XML

Nei paragrafi precedenti, XML è stato presentato come formato per la definizione, attraverso la marcatura, della struttura e della semantica di documenti di testo.
Questa descrizione, benché didatticamente utile e perfettamente coerente con la storia di XML, non rispecchia però lo stato attuale dello standard, che si rivolge alla rappresentazione di qualsiasi tipo di informazione che possa essere scambiata tra sistemi software.

2.1. XML Infoset e i modelli di contenuto

La parte oggi considerata più importante dell'intero standard XML, il cosiddetto *Infoset*, descrive i documenti XML come insiemi di *oggetti astratti*, che hanno una o più proprietà dotate di nomi convenzionali, senza fare alcuna ipotesi sul loro formato di memorizzazione a basso livello (detto anche *formato di serializzazione*) che può essere o meno quello canonico basato sui caratteri.
Anche la modellazione di Infoset segue la struttura gerarchica dei testi (Sottoparagrafi 1.1. e 1.2.); l'intero documento XML costituisce, infatti, un *oggetto-documento* paragonabile alla radice di un albero *multisorte*, cioè composto da oggetti di vario tipo: gli elementi XML, il loro contenuto, le istruzioni di elaborazione ed i commenti. Gli *oggetti-contenuto* (assieme ai commenti e le istruzioni di elaborazione) costituiscono i nodi terminali (le foglie) dell'albero Infoset, mentre gli *oggetti-elementi* fungono da nodi intermedi.
Ogni sistema software che deve manipolare dati XML è libero di definirne una propria rap-

[3] Ora è adottato come standard anche nel progetto TEI.

presentazione interna di basso livello, purché rispetti lo standard Infoset.
Il fatto che il formato prescelto sia quasi sempre quello basato sul testo risolve senza dubbio i problemi di chi (come gli autori di quest'articolo) deve scrivere esempi di dati XML; in molte applicazioni, però, risulta consigliabile usare rappresentazioni XML non testuali, ad esempio, per ridurre le dimensioni di messaggi XML inviati su una rete congestionata o a banda stretta.
Se queste rappresentazioni non testuali rispettano lo standard Infoset, i sistemi che le utilizzano possono interoperare traducendo, quando necessario, l'informazione in un formato di serializzazione canonico basato sul testo.
La parte centrale di Infoset, detta *nucleo* o *XML Information Set Core*, elenca i blocchi e le proprietà fondamentali che devono essere riconosciuti e gestiti da tutte le implementazioni di XML[4].
Spesso, gli elementi di un documento XML non contengono direttamente dei dati, ma servono a racchiudere e a correlare uno o più elementi figlio.
L'esempio, <dati_personali> non contiene dati ma ha lo scopo di tenere insieme altri marcatori, che a loro volta contengono le informazioni anagrafiche sul candidato.
Nella terminologia di Infoset, la struttura interna di un elemento XML viene chiamata *modello di contenuto*; nel caso di <dati_personali>, si parla *di modello di contenuto di soli elementi*, mentre per gli elementi che contengono solo dati e non hanno elementi figlio (come <comune> o <e-mail>) si parla di *modello di contenuto di soli dati*.
Quando si usa XML per rappresentare dati da manipolare via software, questi due modelli di contenuto sono del tutto sufficienti; ma nei sistemi di gestione dei documenti è pratica comune intercalare elementi figlio con dati in forma di caratteri. Questo modello di contenuto, noto come *modello a contenuto misto*, sta per essere progressivamente abbandonato.

[4] È importante notare che la definizione di interfacce per accedere da programma ai dati XML non è parte di Infoset, ma è oggetto di standard separati (Paragrafo 4).

3. XML SCHEMA

Anche se a prima vista può sembrare che le DTD forniscano tutte le funzionalità necessarie per definire la struttura dei documenti XML, ad uno sguardo più attento esse presentano anche numerosi svantaggi, che hanno suggerito l'adozione di soluzioni alternative.
In primo luogo, esiste il problema della mancanza di uniformità tra i *dati* (il contenuto dei documenti XML) e i corrispondenti *metadati*, cioè le DTD usate per descriverne la struttura. In altre parole, poiché la sintassi usata per scrivere le DTD non è essa stessa una marcatura XML valida, non è possibile elaborare i metadati con gli stessi strumenti software usati per manipolare i dati.
Oltre ad essere sintatticamente non uniformi ai dati che descrivono, le DTD sono carenti anche dal punto di vista dei *tipi di dati elementari*, cioè dei valori che possono assumere gli elementi e gli attributi. Nella DTD del curriculum, ad esempio, non sarebbe possibile limitare il contenuto degli elementi <da> e <a> in modo che si possa trattare solo di numeri interi nell'intervallo da 1960 a 2100, anche se questo vincolo risulterebbe prezioso per la validazione dei dati.
In generale, ci si è resi conto che quando si definisce la struttura di dati in formato XML sarebbe utile disporre dello stesso repertorio di tipi di dati elementari incorporati, comunemente a disposizione dei progettisti di database relazionali; ma le DTD richiedono un'obbligata definizione manuale anche di tipi di uso molto comune come le date.
Inoltre, la tecnica con cui sono definiti gli elementi delle DTD (Sottoparagrafo 2.1) non è adatta a complessi progetti di *sistemi di tipi di dati*, in cui possono esistere più tipi con significative parti comuni.
Se nell'esempio fosse stato necessario disporre di due elementi <indirizzo>, uno in formato europeo (<indirizzoEU>) e uno statunitense (<indirizzoUSA>) sarebbe stato indispensabile definirli in maniera del tutto indipendente, benché le differenze siano marginali.
I linguaggi di programmazione a oggetti, risolvono situazioni del genere attraverso la

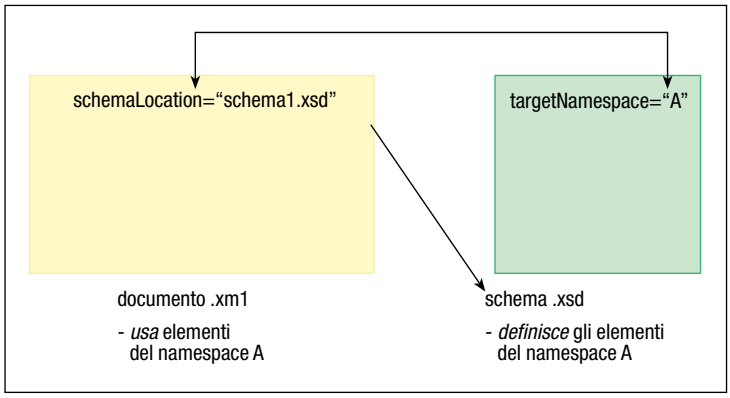


FIGURA 5
Schema e target namespace

fattorizzazione degli elementi comuni a più tipi in appositi *tipi di dati astratti*, a partire dai quali vengono definiti per differenza i tipi di dati concreti effettivamente stanziati, cioè usati, nei programmi. Purtroppo, le DTD XML non permettono di definire elementi per differenza rispetto ad altri.

Lo standard *XML Schema* del W3C è nato proprio per risolvere questi problemi. Si tratta di una sintassi XML che distingue tra *tipi di dati* ed *elementi XML* appartenenti a quei tipi. Oltre ad includere una quarantina di tipi elementari incorporati, XML Schema fornisce al progettista più o meno lo stesso potere espressivo degli attuali linguaggi di programmazione ad oggetti. In altri termini, XML Schema è un completo *Data Definition Language* (DDL) basato su XML che incoraggia (anche se non impone) il classico modo di procedere dei progettisti software: partire dalla *definizione* di tipi di dati per eseguire una successiva *dichiarazione* di elementi XML appartenenti a quei tipi.

Nel modello di XML Schema, le DTD sono del tutto assenti: ogni documento XML fa uso degli elementi definiti da un certo Schema XML. Ciò corrisponde all'orientamento, più volte ribadito dal W3C, di considerare strategica la tecnologia degli Schemi rispetto a quella delle DTD.

Uno schema definisce uno spazio di denominazione, o *target namespace*, che è composto dai nomi dei tipi e degli elementi, appartenenti a quei tipi, definiti al suo interno (Figura 5). Uno schema può utilizzare al suo interno anche elementi o tipi definiti in altri namespace (*source namespace*), *qualificandoli* con un prefisso che corrisponde allo *Uniform Resource Identifier* (URI) che ne identifica la provenienza. Un URI identifica univocamente una risorsa sulla Rete; il tipo maggiormente conosciuto di URI è la URL abitualmente utilizzata per le pagine Web, che oltre a identificare univocamente le pagine permette ai browser Web di recuperarle su Internet. Il prefisso xsd: è solitamente usato per qualificare gli elementi del linguaggio XML Schema stesso, che sono usati per scrivere gli altri schemi e costituiscono lo "schema degli schemi" messo a punto dal W3C.

La figura 6 mostra la tecnica di *validazione in due fasi* dei documenti XML usando gli schemi.

All'interno degli schemi XML, i tipi di dati possono essere definiti per *restrizione* o *estensione* di tipi già esistenti, mentre tra tipi completamente diversi possono essere stabilite sofisticate relazioni di *sostituibilità* (per esempio, è possibile permettere l'interscambiabilità di un tipo di dati "metropolitana" con un tipo di dati "treno" che pure non hanno in comune alcuna parte della loro struttura). La seguente definizione di *tipo complesso* (*ComplexType*) XML Schema specifica la struttura interna di un tipo CurriculumItem:

```
<xsd:complexType name="CurriculumItem
Type">
      <xsd:sequence>
         <xsd:element name="periodo"type=
         "TimeIntervalType"/>
         <xsd:element name="organizzazione"
         type="OrganizationType"/>
      </xsd:sequence>
   </xsd:complexType>
```

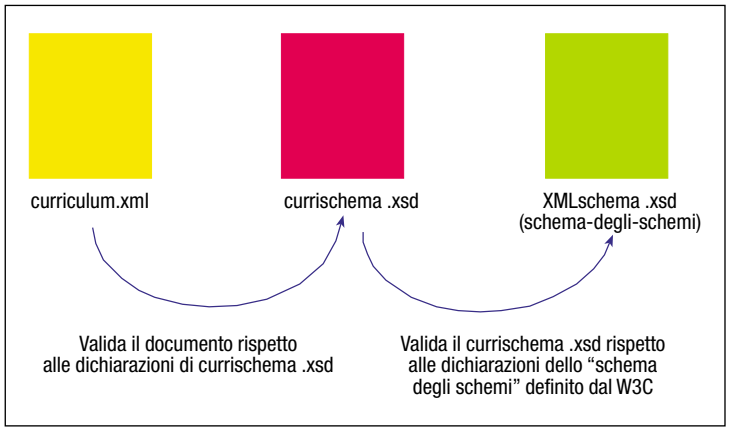
Come si vede, in XML Schema la struttura di un tipo complesso dotato di nome può essere definita in termini di una sequenza di elementi di altri tipi (le cui definizioni sono qui omesse per brevità). Possiamo usare il consueto stile di programmazione a due livelli (definizione di tipo e dichiarazione di variabile) per dichiarare l'elemento <esperienza> in modo che abbia la struttura precedentemente definita. Basterà scrivere:

```
<xsd:element name="esperienza" type=
"CurriculumItemType"/>
```

Ovviamente, è anche possibile dichiarare l'elemento <esperienza> e definirne contestualmente la struttura senza far riferimento ad alcun tipo denominato, come segue:

```
<xsd:element name="esperienza">
      <xsd:sequence>
         <xsd:element name="periodo" type
         ="TimeIntervalType"/>
         <xsd:element name="organizzazione"
         type="OrganizationType"/>
      </xsd:sequence>
   </xsd:element>
```

Ma in questo modo uno dei motivi fondamentali per usare gli schemi viene meno: la struttura interna di <esperienza> non potrà infatti essere ereditata da altri tipi o comunque riutilizzata per definirli.
Lo stile di definizione/dichiarazione a due livelli, appena descritto, costituisce un approccio semplice, ma molto efficace, alla *modellazione di dominio* usando gli Schemi XML.
Questa tecnica consiste nell'usare le definizioni *ComplexType di XML* Schema per progettare il completo repertorio informativo di tipi relativo a un dominio applicativo oppure ad un'organizzazione. Ai tipi di questo Schema principale saranno riferiti gli elementi usati di volta in volta nei documenti o messaggi XML che circolano all'interno dell'organizzazione stessa. Per dichiarare gli elementi XML si usano spesso altri schemi, gestiti indipendentemente da quello principale. Tecnicamente, però, le dichiarazioni di elementi in essi contenute faranno tutte riferimento al *namespace* sorgente dello schema principale (per esempio, definendo e usando un prefisso come orgtypes:) oltre, naturalmente, a riferirsi al namespace di XML Schema, qualificandolo come xsd: .
A livello più basso, lo standard comprende la nozione di tipo semplice o *SimpleType*, che consente la definizione di nuovi tipi per enumerazione o restrizione dei 40 tipi elementari, oppure specificando espressioni regolari. Viene mostrata, di seguito, la definizione di un tipo per l'elemento <da> del nostro esempio, operando una restrizione sul tipo elementare incorporato date.




```
<xs:simpleType name="sinceType">
   <xs:restriction base="xs:date"/>
  </xs:simpleType>
```

FIGURA 6
Validazione di un documento rispetto a uno schema

4. PARSING E VALIDAZIONE DEI DOCUMENTI: I FORMATI SAX E DOM

Il risultato della validazione di un documento XML rispetto a una DTD o ad uno Schema (operazione spesso chiamata *parsing*) è una rappresentazione dell'Infoset del documento in un formato più adatto all'elaborazione da parte di programmi informatici, rispetto al formato di serializzazione.
Le due rappresentazioni più comuni di XML sono il *Document Object Model* (DOM) Level 2 e la *Simple API* (*Application Program*(*ming*) *Interface*) *for XML* (SAX). Entrambe queste tecniche si basano sulla traduzione delle astrazioni di Infoset in un modello a oggetti che evita ai programmatori di dover manipolare direttamente i caratteri del formato di serializzazione di XML. Sia SAX che DOM definiscono un insieme di interfacce che permettono ai programmi di accedere all'insieme delle informazioni XML, ma differiscono in alcuni aspetti fondamentali. SAX traduce l'albero Infoset di un documento XML in una sequenza lineare di eventi. Leggendo un file XML, un parser SAX genera un evento ogni volta che incontra uno degli oggetti Infoset; l'applicazione che deve accedere ai dati gestisce questi eventi con altrettante chiamate di funzione. Un parser DOM traduce l'Infoset del file in un albero di oggetti. DOM è particolarmente adatto alle applicazioni che devono rappresentare interamente in memoria un docu-

mento XML, mentre le applicazioni basate su SAX non hanno bisogno di tenere in memoria l'intera rappresentazione dell'Infoset.
Una seconda non trascurabile differenza deriva dal fatto che DOM è una Raccomandazione del W3C e ha dietro di sé il peso istituzionale di questa organizzazione, mentre SAX è uno standard di fatto sviluppato da un gruppo di progettisti che facevano capo alla mailing list XML-DEV, supervisionato da David Megginson. La mancanza di un'approvazione "ufficiale" di SAX non ha tuttavia impedito che la maggior parte dei prodotti software per XML supporti sia SAX che DOM. Poiché sia il formato SAX che DOM sono conformi a Infoset, possono interoperare senza problemi (infatti, molte implementazioni di DOM sono state scritte sulla base di codice SAX).

5. SELEZIONE DI INFORMAZIONI XML: LO STANDARD XPATH

La struttura gerarchica di Infoset può essere usata per individuare sottoinsiemi dei nodi di un documento attraverso un semplice linguaggio che descrive l'attraversamento dell'albero. In questo modo, invece di scrivere le routine di estrazione e di attraversamento in un linguaggio di programmazione, gli sviluppatori possono ricorrere a semplici espressioni e lasciar fare tutto il lavoro al parser XML.
Il linguaggio proposto dal W3C per estrarre sottoinsiemi di nodi dell'albero dei documenti XML è chiamato *XPath* (*XML Path Language 1.0*).
Il costrutto più importante usato nelle espressioni XPath è chiamato *percorso di posizionamento*. Proprio come i percorsi dei file system, i percorsi di posizionamento XPath possono essere *assoluti* o *relativi*; quelli assoluti iniziano con una barra obliqua (/) e indicano che la navigazione deve incominciare dal nodo radice nel modello ad albero. Un percorso di posizionamento relativo non inizia con una barra obliqua: ciò significa che l'attraversamento dell'albero deve iniziare dal nodo con cui inizia il percorso di posizionamento.
Ecco l'espressione XPath che individua tutti gli elementi <nome> che sono figli di elementi <dati_personali>, i quali a loro volta discendono dal nodo radice <curriculum> :

/curriculum/dati_personali/nome

Nel caso di questo semplice esempio, non sarebbe difficile scrivere un programma che esegue lo stesso compito usando le API SAX o DOM, ma nel caso di espressioni più complicate che includano caratteri jolly e condizioni logiche, aumentano i vantaggi di delegare il codice di attraversamento al parser XML. L'espressione che segue estrae i nodi Infoset <località> discendenti dalla radice <curriculum> e dotati di un attributo tipo il cui contenuto sia piazza.

curriculum/*/località/[@tipo='piazza']

Oltre ad essere utilizzato direttamente XPath viene presupposto dallo standard XSLT 1.0 per le trasformazioni di documenti XML e dal nuovo linguaggio di interrogazione XQuery proposto dal W3C. XSLT usa XPath per identificare i sottoinsiemi di nodi di un documento di origine che saranno tradotti in parti del documento di output.

6. I LINGUAGGI DI STILE: XSL E LE REGOLE DI TRASFORMAZIONE

Nel sottoparagrafo 1.1 è stato brevemente introdotto XSL, come standard per la presentazione dei documenti XML, per visualizzarli cioè su schermo, su carta o riprodurli via audio. Nella definizione di questo standard divenne chiaro che si trattava di un processo in due fasi: prima vi fu una trasformazione strutturale, e poi subentrò il processo di formattazione vero e proprio. Per questo, il linguaggio XSL fu diviso in: XSL-T per definire le trasformazioni e XSL-FO (*XSL Formatting Objects*) per definire la fase di formattazione. XSL-FO è un namespace XML in cui i marcatori descrivono aree della schermata o della pagina stampata e le loro proprietà. Poiché si tratta di marcatori XML, XSL-T non ha problemi per generarli come suo output. La definizione di XSL-FO è però un'impresa di difficoltà paragonabile alla creazione dello standard PostScript e i prodotti che implementano XSL-FO sono in una fase, ancora primitiva, di sviluppo. Per questo, i documenti XML vengono di solito tradotti usando XSLT per produrre output in HTML, visualizzabili attraverso un browser standard per la navigazione del Web.

6.1. Introduzione a XSLT

XSLT è un linguaggio standard progettato per trasformare documenti XML. Le sue applicazioni sono molteplici: per esempio, le organizzazioni che hanno investito nella creazione di fonti d'informazioni in formato XML standard devono essere in grado di trasformarle per inviarle non solo al tradizionale browser web, ma anche ai terminali mobili e ai televisori. Inoltre, può essere necessario trasformare dati XML, inviati come messaggio, in modo che rispettino la DTD o lo Schema adottati dal destinatario. Poiché il commercio elettronico interaziendale è sempre più diffuso, la quantità di dati scambiati tra aziende aumenta giornalmente e questa esigenza diventa sempre più pressante.

Quando XSLT non era ancora disponibile, per eseguire queste trasformazioni era necessario scrivere applicazioni software in linguaggi complessi come C++ o Java: con XSLT diventa possibile (e molto più comodo) descrivere le trasformazioni desiderate usando un *linguaggio dichiarativo* che esprima le trasformazioni come una serie di regole che definiscano l'output che deve essere prodotto. Inoltre, XSLT si basa su un parser DOM per convertire il documento XML nella struttura ad albero corrispondente: è l'Infoset del documento che XSLT manipola, non il documento stesso. Un foglio di stile XSLT è un programma dichiarativo che contiene un insieme di regole che permettono di navigare nell'albero Infoset, che rappresenta un documento, di scegliere nodi specifici e di eseguire elaborazioni anche complesse su questi nodi.

6.2. Modalità di trasformazione

L'elaborazione di un foglio di stile XSLT consiste di due fasi distinte:

- la *trasformazione strutturale*: i dati XML in ingresso sono convertiti in una struttura che riflette l'output desiderato;
- la *formattazione*: la nuova struttura è convertita nel formato di uscita richiesto, per esempio HTML o PDF.

La trasformazione strutturale consiste in operazioni come la selezione dei dati (cioè la scelta di alcuni degli elementi e attributi nel documento XML in ingresso), il loro ordinamento e l'esecuzione di conversioni aritmetiche (per esempio la trasformazione di centimetri in pollici). La figura 7 mostra il flusso di dati della trasformazione XSLT.

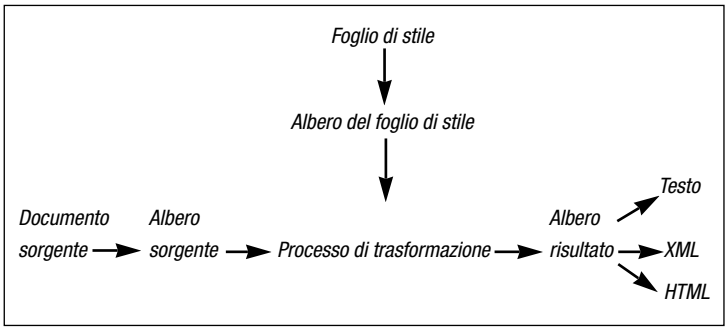


FIGURA 7
Flusso di dati della trasformazione

6.3. XSLT e i linguaggi d'interrogazione per basi di dati

XSLT presenta alcune somiglianze con i linguaggi di interrogazione usati per i database relazionali. In un database relazionale, i dati consistono in una serie di tabelle; usando il linguaggio di interrogazione SQL (*Structured Query Language*), è facile definire operazioni di estrazione e aggregazione dei dati delle tabelle, per poi convertire il risultato in HTML. Come si vedrà in seguito, molti database relazionali oggi consentono di estrarre informazioni anche in formato XML; ma che vantaggio c'è nel delegare a XSLT i compiti di trasformazione su queste informazioni? La risposta sta nel fatto che le trasformazioni XSLT sono adatte anche a *sorgenti di dati eterogenee*, perché funzionano per qualunque sorgente di dati che esporti XML e non solo per i database relazionali. Naturalmente, XSLT e SQL sono destinati a coesistere: i dati saranno memorizzati in database relazionali e trasmessi tra i sistemi in formato XML, personalizzandoli a seconda delle esigenze del ricevente.

6.4. La sintassi di XSLT

In genere, un linguaggio di trasformazione definisce una sintassi per selezionare i dati che devono essere elaborati, siano essi memorizzati in un database relazionale o in un documento XML. Per questo scopo in SQL vi è la clausola SELECT; in XSLT l'equivalente sono le espressioni XPath di cui ci si è occupati in precedenza (Paragrafo 5).

I percorsi di attraversamento di XPath sono

usati per identificare i nodi desiderati all'interno di un documento XML, in base ad un percorso attraverso il documento XML (partendo dalla radice o dalla posizione corrente) o in base al contesto in cui il nodo appare. XSLT viene usato poi per manipolare i risultati di queste selezioni (modificando i nodi selezionati, costruendo nuovi nodi e così via).
Un'altra importante proprietà concettuale comune a XSLT e SQL è la *chiusura*, la proprietà per cui l'output di una trasformazione ha la stessa struttura dati dell'input. In SQL questa struttura (cioè, l'ambiente rispetto a cui il linguaggio è chiuso) è definita dalle tabelle relazionali, nel caso di XSLT si tratta degli alberi Infoset dei documenti XML.
La proprietà di chiusura implica che le operazioni eseguite usando il linguaggio possono essere *composte* per definire operazioni molto più complesse: si può prendere l'output di un'operazione e renderlo l'input dell'operazione successiva. In SQL è possibile eseguire ciò definendo viste o interrogazioni secondarie (sub-query), mentre in XSLT è possibile filtrare i dati d'ingresso attraverso una serie di fogli di stile.

6.5. Esempi di trasformazioni

Il seguente frammento di foglio di stile mostra una versione semplificata della trasformazione necessaria per tradurre, in HTML, il nostro curriculum di esempio:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
"http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/
Format">

<xsl:template match="curriculum">
<HTML>
    <HEAD>
        <TITLE>
          Curriculum vitae
        </TITLE>
    </HEAD>
    <BODY>
    <xsl:apply-templates/>
    </BODY>
</HTML>
</xsl:template>

<xsl:template match="dati_personali">
   <H1>Riepilogo dati personali</H1>
   <TABLE Border="1" width="100%">
        <TR>
           <TD>nome</TD>
           <TD>cognome</TD>
           <TD>via</TD>
           <TD>civico</TD>
           <TD>cap</TD>
           <TD>comune</TD>
           <TD>provincia</TD>
           <TD>mail</TD>
           <TD>tel</TD>
           <TD>cellulare</TD>
           <TD>fax</TD>
           <TD>datanascita</TD>
           <TD>comunenascita</TD>
           <TD>provincianascita</TD>
           <TD>cittadinanza</TD>
           <TD>statocivile</TD>
        </TR>
<xsl:apply-templates/>
  </TABLE>
  <br/>
</xsl:template>

<xsl:template match="nome">
        <TD><xsl:apply-templates/></TD>
        </xsl:template>

<xsl:template match="cognome">
        <TD><xsl:apply-templates/></TD>
         </xsl:template>
.....
</xsl:stylesheet>
```

È interessante notare che il foglio di stile XSLT è esso stesso un documento XML. Non potendo trattare qui in modo esauriente il linguaggio di trasformazione[5], ci si limiterà a fare qualche commento, seguendo passo per passo l'esempio proposto. Il foglio di trasformazione inizia con:

```
<xsl:stylesheet version="1.0"xmlns:xsl=
"http://www.w3.org/2000/XSL/Transform"
xmlns:fo="http://www.w3.org/2000/XSL/
Format">
```

[5] Il file del foglio di stile è comunque disponibile così come gli altri esempi menzionati nell'articolo alla url http://beserker.crema.unimi.it/xml

Si tratta dell'intestazione standard XSLT. L'attributo xmlns:xsl è una dichiarazione di XML Namespace, che indica che verrà usato il prefisso xsl per contrassegnare i nomi degli elementi definiti nello schema dello standard XSLT rilasciato dal W3C, in modo da non confonderli con gli altri elementi utilizzati nel documento di input e definiti in uno Schema o DTD dell'utente.

<xsl:template match="curriculum">

Un elemento <xsl:template> definisce il modello da cercare nel documento sorgente. L'attributo match="curriculum" indica che la regola viene applicata all'elemento-radice del documento XML sorgente, <curriculum>. È importante rilevare che qui, curriculum, non è un nome di tag, ma un'espressione di attraversamento Xpath, che identifica un insieme di nodi (composto in questo caso dal solo nodo radice del documento d'esempio). Una volta che la condizione di applicazione della regola è stata selezionata, il corpo della regola dice al foglio di stile quale output generare. Per questa prima regola, si tratta di una sequenza di elementi HTML (compresa l'intestazione di una tabella) che deve essere copiata nel file di output. Segue poi un elemento cruciale, <xsl:apply-templates/>, che causa una chiamata ricorsiva dell'intero foglio di stile, all'insieme cioè delle regole XSLT qui contenute, a cui viene dato in input il sottoalbero del documento originale che corrisponde al punto fin qui raggiunto.
Il foglio prosegue, quindi, elaborando i nodi figli di curriculum e seguendo sue semplici strategie: nel caso di nodi interni (con modello di contenuto a soli elementi) si limita ad eseguire una chiamata ricorsiva, mentre, nel caso di nodi foglia, aggiunge una semplice formattazione HTML e i marcatori <TD> necessari per aggiungere una riga alla tabella HTML. La figura 8 mostra il risultato dell'applicazione del foglio di stile al documento d'esempio.
Benché questo semplice esempio ha riguardato la sola traduzione in HTML, a tecnica di trasformazione dichiarativa basata su XSLT presentata in questo paragrafo, si presta molto bene alla realizzazione di *servizi informativi multi protocollo*, in grado di rispondere alle richieste degli utenti trasformando sul momento dati in formato XML per ottenere documenti formattati nel modo più adatto al profilo d'interessi dell'utente e soprattutto al tipo di terminale che sta utilizzando.

FIGURA 8
Il risultato dell'applicazione del foglio di stile al nostro curriculum d'esempio

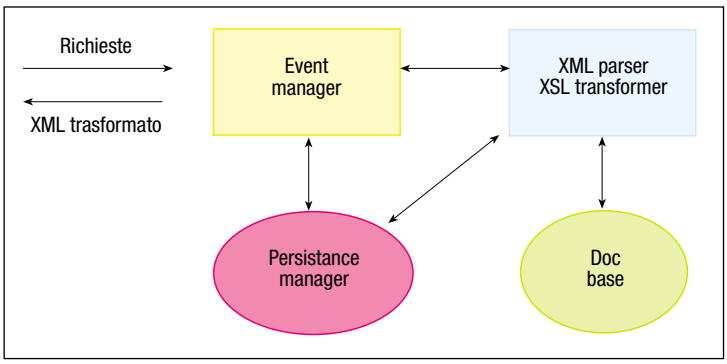


FIGURA 9
Architettura con trasformazione XSL lato server

7. APPLICAZIONI

7.1. Editoria

Una delle tante potenzialità delle tecnologie XML e XSLT, tra loro combinate, nella pubblicazione dinamica dei contenuti, è l'utilizzo del codice XSL parametrico per la realizzazione di servizi multiprotocollo. Ma XML, per la sua capacità di rappresentare la struttura logica di un documento in modo distinto rispetto alla modalità di rappresentazione grafica, ha molte altre utili applicazioni in campo editoriale. Con XML diventa infatti possibile il *riutilizzo di uno stesso documento codificato in XML per diversi tipi di stampa* (per esempio in bianco e nero o in quadricromia), *diversi formati* (HTML, PFD, PostScript) e *diverse modalità di pubblicazione* (carta, CD-ROM, World Wide Web), senza dover apportare alcuna modifica alla codifica XML, ma semplicemente cambiando i fogli di stile associati.
XML facilita inoltre la gestione delle *riedizioni* e, grazie alle potenzialità di XSLT, rende possibile la *personalizzazione delle edizioni*, in base alle esigenze del committente: è possibile, per esempio, da una collezione di opere estrarre facilmente un'antologia.

7.2. Interoperabilità e integrazione

Quasi tutti i principali produttori di software hanno incorporato nel progetto delle loro piattaforme di servizi basate sul Web la nozione di *trasformazione in linea* di dati XML.
La figura 9 rappresenta una classica soluzione strutturale di questo tipo, che prevede una *base documentale* (che può essere sostituita da un'interfaccia verso un database relazionale) da cui l'informazione XML viene prelevata all'atto di una richiesta. I dati XML vengono sottoposti a parsing per ricavare l'albero DOM, su cui si applica poi la *trasformazione XSL lato server* necessaria per rispondere alla richiesta remota. I nodi dell'albero DOM selezionati vengono resi *persistenti* e salvati in una memoria cache (detta anche *persistence manager*) per essere recuperati più velocemente nel caso in cui arrivino altre richieste che li riguardano.
Questa visione trasformazionale dei server Web si è recentemente evoluta nell'idea dei *Web services*, servizi remoti che ricevono *richieste* o *invocazioni* sotto forma di messaggi XML e inviano risposte parimenti codificate come XML.
Un campo applicativo di particolare interesse per l'integrazione via XML è quello relativo all'accesso via Internet ai database aziendali e alle operazioni di commercio elettronico.
Le principali applicazioni riguardano lo scambio di dati tra varie attività aziendali nell'ambito di una catena di approvvigionamento (*Electronic Data Interchange* o EDI), secondo il ben noto modello del commercio elettronico B2B (Business to Business), e nelle transazioni su Internet che coinvolgono clienti finali (commercio elettronico Business to Consumer o B2C).
La figura 10 mostra una tipica transazione interaziendale basata sull'interscambio di documenti in formato XML.
La tecnologia XML viene sfruttata a due livelli ben distinti tra loro. In primo luogo, la comunicazione tra applicazioni software indispensabile per queste applicazioni è stata ottenuta mediante tecnologie software d'integrazione basate su protocolli proprietari come Microsoft COM/DCOM (*Component Object Model/Distribuited* COM) o standard come CORBA (*Common Object Request Broker Architecture*). I protocolli "leggeri" per l'esecuzione remota di servizi attraverso HTTP (*Hyper Text Transfer Protocol*) e XML hanno guadagnato consensi tra gli sviluppatori di servizi interoperabili per il loro costo computazionale limitato e per la loro flessibilità.
I protocolli "leggeri" basati su XML come SOAP (*Simple Object Access Protocol*), che è alla base dell'architettura Microsoft .NET, permettono di eseguire l'integrazione tra applicazioni remote attraverso l'interscambio di dati standardizzati in formato XML.

7.3. Semantic Web e RDF

Uno degli aspetti più interessanti della marcatura XML è la possibilità di usarla per denotare la semantica dei dati, rendendo così più facile la loro selezione e classificazione.
Le usuali pagine Web non contengono alcuna esplicita rappresentazione del loro significato, a parte quella fornita dai marcatori HML <META>, usata dai motori di ricerca per l'indicizzazione delle pagine.
Questa situazione rappresenta un problema soprattutto per chi deve scrivere degli agenti software in grado di raccogliere ed elaborare senza intervento dell'utente l'informazione presente sui siti Web della Rete. I metadati XML (sotto forma, per esempio, di XML Schema) offrono solo una soluzione molto parziale a questo problema, prima di tutto perché, come abbiamo visto, la semantica dei marcatori definiti dal namespace XML di un'organizzazione è tutt'altro che ovvia per altre organizzazioni. In secondo luogo, uno Schema XML descrive soltanto dati in formato XML nativo. Questo apparente vicolo cieco tecnologico è stato affrontato dal W3C proponendo uno standard internazionale basato su XML per la rappresentazione astratta dei contenuti delle risorse di rete, siano o meno espressi in XML, che può essere usato dagli agenti software per le loro ricerche. Questo standard prende il nome di *Resource Description Format* o RDF.

7.4. RDF in breve

Il concetto principale su cui si basa lo standard RDF è che i dati sono descritti da asserzioni, relative a risorse identificate in modo univoco tramite un *Uniform Resource Identifier* o URI[6] (file XML o sottoalberi individuati da un XPath, pagine HTML, interi siti Web). Le asserzioni RDF specificano una proprietà della risorsa (per esempio il titolo, la data di creazione o il tipo) e un valore, che può essere una stringa, numero, un frammento XML o un'altra risorsa.
L'asserzione RDF che segue, specifica che

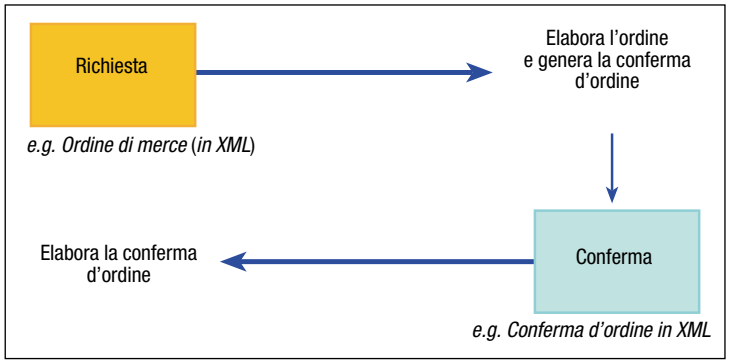


FIGURA 10
Transazione interaziendale via Rete con scambio di documenti XML

l'autore della risorsa del nostro curriculum d'esempio, si chiama Carlo Bianchi:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:statement xmlns:rdf=
"http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdfs=
"http://www.w3.org/2000/01/rdf-schema#">
    <rdf:subject resource="http://www.cercalavoro.org/Curricula/curriculum.xml"/>
    <rdf:predicate resource="http://www.purl.org/dublin-core#author"/>
    <rdf:object>Carlo Bianchi</rdf:object>
</rdf:statement>
```

La cosa più interessante di questa sintassi RDF, che per il resto dovrebbe essere largamente auto-esplicativa, è l'associazione esplicita (eseguita mediante concatenazione di stringhe) stabilita dal marcatore <rdf:predicate>.
Questa associazione lega il nome della proprietà (author) e la URI che contiene la definizione di tutte le proprietà utilizzabili per la descrizione e delle relazioni tra loro (in questo caso si fa riferimento allo standard *Dublin Core*, un ben noto standard di biblioteconomia che elenca le principali proprietà che può avere un testo scritto).
In generale, l'elenco delle proprietà predicabili in RDF sulle risorse di un certo dominio applicativo e le relazioni che esistono tra queste proprietà costituiscono un modello concettuale del dominio stesso, di potere espressivo più o meno equivalente a quello di una *rete semantica*. Il linguaggio che si usa per specificare proprietà e relazioni tra loro è anch'esso basato su XML e prende il nome di RDF Schema (RDFS). Gli schemi RDFS, su cui non è il caso di soffermarsi, organizzano i concetti del dominio in

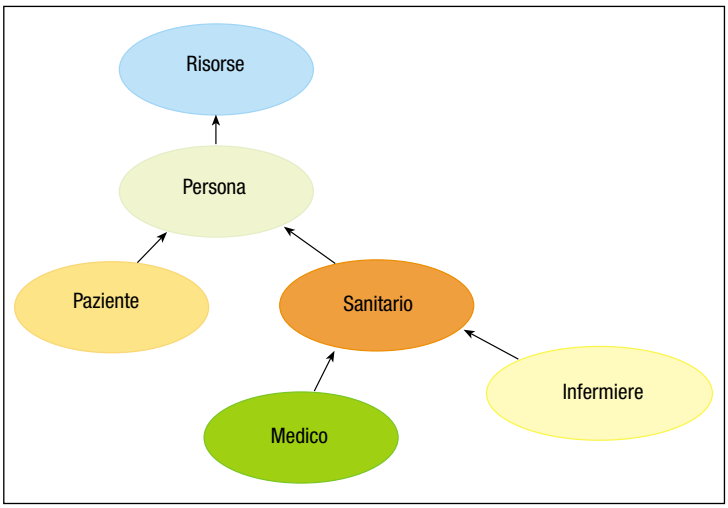
[6] Un indirizzo logico per l'identificazione di risorse sul Web, analogo all'URL, di cui è una generalizzazione.

modo elaborabile da agenti software e costituiscono un primo passo verso l'elaborazione di *ontologie di dominio* e verso l'indicizzazione delle risorse del dominio stesso usando questi concetti piuttosto che semplici parole-chiave[7].
Ecco un frammento di uno schema RDF che esprime il fatto che un Operatore Sanitario è una sottoclasse di Persona:

```
<?xml version="1.0" encoding="UTF-8"?>
    <rdf:RDF xmlns:rdf="http://www.w3.
org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/
rdf-schema#">
    <rdfs:Class rdf:ID="Operatore Sanitario">
    <rdfs:subClassOf rdf:resource="#Persona"/>
    </rdfs:Class>
    </rdf:RDF>
```

La corrispondente gerarchia di concetti, che risulterà familiare a chi conosce la programmazione orientata agli oggetti, è riportata nella figura 11.
In realtà RDFS ha un potere espressivo analogo a quello dei Data Definition Language dei database ad oggetti, ed è ben più limitato di altri linguaggi esistenti per la descrizione di ontologie. Esistono numerose proposte per la sua estensione, soprattutto da parte di chi si occupa di sistemi per la gestione della conoscenza.

FIGURA 11
La gerarchia d'ereditarietà a cui si riferisce lo schema RDFS d'esempio



8. STRUMENTI SOFTWARE

8.1. Software di base

Per la diffusione di XML molto importante è la disponibilità gratuita di *browser XML*, cioè di programmi che consentono la semplice lettura di documenti XML. Oltre ai molti browser Open Source (per lo più scritti in Java), tra quelli commerciali segnaliamo in particolare Microsoft Internet Explorer e Netscape 6.
Per ragioni di ottimizzazione della velocità di visualizzazione dei documenti XML, non sono invece generalmente inclusi nei browser *parser di validazione* del documento rispetto alla DTD o allo Schema associato. Parser di validazione e XSLT sono disponibili come prodotti Open Source, a se stanti, o come componenti di complesse piattaforme software lato server (sottoparagrafo 7.2.): l'elaborazione del *parsing* è infatti molto più veloce se effettuata sul lato server, piuttosto che da browser o altri prodotti sul lato client.
I file XML, essendo file di testo, possono essere editati e modificati con comuni editor ASCII. Tuttavia, per facilitare il compito ai redattori di documenti XML, sono nati specifici *editor XML* che, pur garantendo all'utente il controllo e la visibilità sulla codifica XML, forniscono utili funzionalità, quali l'evidenziazione con colori diversi dei tag XML, il controllo sulla coerenza sintattica (cioè sull'essere il documento ben formato), la validazione rispetto alla DTD o allo XML Schema di riferimento e la visualizzazione della struttura ad albero degli elementi XML.
Tra i vari editor in commercio (ricordiamo qui, tra gli altri, Microsoft XML Notepad, XML Pro, CLIP! XML Editor, CUESoft EXml, XMetal), molto popolare qui in Italia è XML Spy, distribuito da una software house austriaca.
Questi editor sono professionali e si rivolgono ad utenti esperti, competenti di informatica. Un territorio ancora relativamente inesplorato è quello dei cosiddetti *editor visuali* per XML, che semplificano al massimo la gestione dei tag, avvicinando XML agli utenti di base, non specialisti[8].

[7] Lo scenario, proposto dal W3C, secondo cui le risorse del WWW saranno indicizzate mediante concetti presi da ontologie elaborate in modo consortile prende il nome di *Semantic Web Initiative*.

[8] Un editor sperimentale di questo tipo (Visual XML Editor) è stato recentemente prodotto come tesi di laurea al Dipartimento di Tecnologie dell'Informazione dell'Università degli Studi di Milano – Polo di Crema.

8.2. XML e database

Molti prodotti per la gestione di basi di dati relazionali sono oggi in grado di esportare in formato XML il risultato delle interrogazioni. La codifica di una tabella relazionale in XML è un'operazione molto semplice, anche se non esiste uno standard preciso per farlo.
L'esempio che segue riporta la tecnica di codifica usata dai database Oracle (il semplice schema XML che esprime la struttura di questa codifica in XML dei dati relazionali è omesso per brevità):

```
<?xml version="1.0"?>
<ROOTDOC>
<DBROW id="1"> <EMPNO>7876</EMPNO>
<ENAME>BIANCHI</ENAME> </DBROW>
<DBROW id="2"> <EMPNO>7499</EMPNO>
<ENAME>ROSSI</ENAME> </DBROW>
</ROOTDOC>
```

La codifica in XML del risultato di un'interrogazione di una base di dati può essere richiesta utilizzando un apposito parametro dell'ambiente applicativo attraverso il quale si colloquia con il database oppure attraverso clausole che estendono (in modo non standard) il linguaggio SQL comunemente usato per interrogare le basi di dati.
Il database Microsoft SQL Server, ad esempio, permette di richiedere la codifica in XML del risultato di una qualsiasi interrogazione, semplicemente aggiungendo all'interrogazione stessa la clausola FOR XML.
Ovviamente, è possibile (anzi, è consigliabile) usare un foglio di stile XSL per passare dal documento XML puro e semplice restituito dal database relazionale a un documento più strutturato, composto da elementi che rispettano il repertorio di tipi aziendali espresso da uno Schema; sono disponibili vari strumenti visuali che possono aiutare il programmatore in questa operazione.
L'aggiornamento di un database relazionale partendo da dati in formato XML è invece un'operazione meno agevole soprattutto perché il modello dei dati gerarchico dell'Infoset XML deve essere "appiattito" (un'operazione chiamata *flattening*) prima che un documento XML possa essere usato per aggiornare una tabella.
Quasi tutti i database relazionali offrono a questo scopo dei programmi d'utilità che visitano l'albero DOM dei messaggi XML in ingresso per generare gli aggiornamenti alle tabelle relazionali; le prestazioni di strumenti di questo tipo in ambienti ad alta intensità di aggiornamenti sono ancora oggetto di investigazione.
Completamente diversa è la natura dei prodotti *database XML nativi*. Si tratta di sistemi che memorizzano, indicizzano e recuperano informazioni usando direttamente XML Infoset come modello dei dati. Questi sistemi, le cui tecniche di gestione devono parecchio a quelle in uso per i vecchi database gerarchici, trovano applicazione soprattutto in ambienti aziendali dove i flussi informativi sono fortemente orientati ai documenti e hanno sorgenti eterogenee (società di consulenza, redazioni, editoria etc.).

Ringraziamenti

Gli autori desiderano ringraziare Gianni Degli Antoni, fonte di preziosi suggerimenti e stimolanti scambi di idee su XML e sulle sue applicazioni, e Dino Buzzetti, per il suo interessante contributo a una teoria del markup, nell'ambito di una più generale teoria del testo.

Bibliografia

[1] Biztalk: http://www.biztalk.org

[2] Box D, Lam A, Skinnard A: *XML: Beyond Markup, Addison-Wesley pubblicato in Italia con il titolo XML: Oltre il Markup*.

[3] Xmlbooks: http://www.xmlbooks.com/

[4] Xmledi: http://www.xmledi.com

[5] Xmlsoftware: http://www.xmlsoftware.com

[6] W3C: www.w3.org

ERNESTO DAMIANI è Professore Associato presso il Dipartimento di Tecnologie dell'Informazione dell'Università di Milano. È stato Visiting Professor presso la George Mason University, VA, US e presso la LaTrobe University, Melbourne, Australia. Tra i suoi interessi di ricerca vi sono i formati semi-strutturati basati su XML per la rappresentazione dell'informazione e il soft-computing.
e-mail: damiani@dti.unimi.it

PAOLO FEZZI laureato in Lettere, è in AICA Responsabile della Qualità nel Progetto ECDL. Ha collaborato con il Dipartimento di Filosofia dell'Università degli Studi di Bologna in progetti di digitalizzazione e codifica XML di testi di storia della scienza.
Ha scritto articoli come redattore di Sistemi e Impresa e contributi per De Agostini e Motta.
e-mail: fezzi@aicanet.it