XML e basi di dati

Angelo Montanari

Dipartimento di Matematica e Informatica
Università di Udine

Cos'è XML

XML è l'acronimo di eXtensible Markup Language

Caratteristiche fondamentali:

(1) XML è un linguaggio formale. XML è definito mediante un insieme di regole formali (una grammatica) che definisce le modalità di generazione di un documento XML.

(2) XML consente la descrizione (markup) dei dati. I dati sono inclusi nei documenti XML come stringhe di testo racchiuse tra opportune etichette (markup tag) che ne descrivono struttura e tipo di contenuto.

(3) XML è estendibile. Il linguaggio mette a disposizione un insieme estendibile di etichette di markup che possono essere adattate ad esigenze di diverso tipo (a differenza di HTML, dove l'insieme di etichette è fisso).

Cosa non è XML - 1

Un documento XML è un documento di testo che può essere letto e modificato con un qualsiasi text editor.

Pertanto, XML **non è**:

- un **linguaggio di presentazione** dei dati come HTML. XML markup definisce il significato (contenuto) dei dati, ma non lo stile di presentazione. Tale informazione può essere inclusa separatamente in un foglio di stile. Fogli di stile scritti nel linguaggio XSL (XML Stylesheet Language) possono essere usati per tradurre dati XML in HTML. Le pagine HTML risultanti possono essere visualizzate con gli usuali browser.

Cosa non è XML - 2

- un **linguaggio di programmazione** come Java. Un documento XML non computa alcunché.
- un **protocollo per il trasferimento dati via rete** come HTTP. XML non trasferisce dati attraverso la rete.
- un **DBMS** come Oracle. XML non memorizza né restituisce dati.

Elementi XML

Un documento XML è un testo racchiuso fra tag di testo. I tag di testo sono racchiusi tra parentesi angolari.

Esempio di documento XML:

$< person >$
$\quad Alan\ Turing$
$< /person >$

Il documento contiene un unico **elemento** denominato *persona*. Tale elemento è racchiuso fra il tag iniziale $< person >$ e il tag finale $< /person >$. I tag sono una forma di markup del testo. Ciò che viene racchiuso tra i tag iniziale e finale è detto **contenuto dell'elemento** (nell'esempio, la stringa $Alan\ Turing$). Un elemento XML può contenere testo libero (detto *character data*) o altri elementi XML.

Un esempio di documento

Esempio.

$< person >$

$< name >$

$< first > Alan < /first >$

$< last > Turing < /last >$

$< /name >$

$< profession > computer\ scientist < /profession >$

$< profession > mathematician < /profession >$

$< profession > cryptographer < /profession >$

$< /person >$

L'elemento *persona* contiene un sottoelemento *nome* e tre sottoelementi *professione*. L'elemento nome, a sua volta, contiene due sottoelementi *nome proprio* e *cognome*.

Caratteristiche di un documento XML

- **Gli elementi non possono sovrapporsi**: se il tag iniziale di un elemento B segue il tag iniziale di un elemento A, il tag finale dell'elemento B deve precedere il tag iniziale dell'elemento A.
- Deve esistere un singolo elemento che racchiude tutti gli altri (**document element**).
- **Ogni elemento**, ad eccezione del document element, **è racchiuso** (in modo diretto) **in esattamente un altro elemento**.

Conseguenza: un documento XML si può rappresentare mediante una **struttura ad albero**: gli elementi sono i **nodi**, il document element è la **radice** dell'albero, i sottoelementi sono i nodi **figli**.

Un altro esempio

Gli elementi XML possono **mescolare testo libero** e **sottoelementi** come nel seguente esempio:

$< person >$
$< first > Alan < /first >< last > Turing < /last > is\ mainly\ known$
$as\ a < profession > computer\ scientist < /profession > .\ However,\ he$
$was\ also\ an\ accomplished < profession > mathematician < /profession >$
$and\ a < profession > cryptographer < /profession > .$
$< /person >$

Vi possono essere anche elementi privi di contenuto (**elementi vuoti**). Esempio: un elemento vuoto indirizzo si può rappresentare come $< address/ >$).

Nota bene: XML è **case sensitive** (*address* and *Address* sono etichette diverse).

Attributi XML

Gli elementi XML possono avere degli **attributi**, che ne descrivono delle proprietà. Un attributo ha la sintassi $nome = "valore"$, dove *nome* è il nome dell'attributo e *valore* una stringa. Le virgolette che racchiudono il valore possono essere singole o doppie. Un elemento può avere un numero arbitrario di attributi, che devono avere nomi diversi. L'ordine degli attributi è irrilevante (a differenza di quello degli elementi).

$< person\ born = "23/06/1912"\ died = "07/06/1954" >$
$< name >$
$< first > Alan < /first >$
$< last > Turing < /last >$
$< /name >$
$< profession > computer\ scientist < /profession >$
$< profession > mathematician < /profession >$
$< profession > cryptographer < /profession >$
$< /person >$

Elementi XML vs. attributi XML

Alcune informazioni possono essere codificate **sia** come valore di **attributi sia** come contenuto di **elementi**.

$< person\ born = \text{"}23/06/1912\text{"}\ died = \text{"}07/06/1954\text{"} >$
$< name\ first = \text{"}Alan\text{"}\ last = \text{"}Turing\text{"} / >$
$< profession\ value = \text{"}computer\ scientist\text{"} / >$
$< profession\ value = \text{"}mathematician\text{"} / >$
$< profession\ value = \text{"}cryptographer\text{"} / >$
$< /person >$

Come scegliere? Attributi per metadati relativi agli elementi, elementi per la codifica dell'informazione.

L'utilizzo delle referenze XML - 1

In tutti gli esempi precedenti, i documenti XML hanno una **struttura ad albero**. Mostriamo ora come XML disponga di un meccanismo per la definizione e l'uso di referenze il cui impiego produce documenti con una **struttura a grafo**.

XML consente di associare un **identificatore** univoco agli elementi come valore di un certo attributo (assumiamo che tale attributo sia denominato *id*).

$< state\ id = "s1" >$
$\quad < scode > CA < /scode >$
$\quad < sname > California < /sname >$
$< /state >$

L'utilizzo delle referenze XML - 2

Per far riferimento all'elemento *state* definito in precedenza, si può utilizzare l'attributo *idref*.

$< city\ id = "c1" >$
$\quad < ccode > LA < /ccode >$
$\quad < cname > Los\ Angeles < /cname >$
$\quad < state\text{-}of\ idref = "s1" / >$
$< /city >$

Si noti come *state-of* sia un elemento vuoto, la cui funzione è di consentire l'accesso ad un altro elemento attraverso il valore dell'attributo *idref*. Mediante le referenze XML, è possibile rappresentare **strutture dati cicliche/ricorsive**. Nell'esempio, si può aggiungere un sottoelemento *capital* all'elemento *state*, con associato l'attributo *idref* che fa riferimento all'elemento *city*.

XML parser

Un **parser per XML** è un software che legge un documento XML e stabilisce se è o meno ben formato. Un documento ben formato rispetta le regole grammaticali di XML:

1. ogni tag iniziale deve avere un corrispondente tag finale;
2. gli elementi non possono sovrapporsi;
3. deve esserci esattamente un document element;
4. i valori degli attributi devo essere racchiusi tra virgolette;
5. i nomi di attributi associati allo stesso elemento devono essere diversi.

Il modo più semplice per effettuare il parsing di un elemento XML è di caricarlo in un web browser che conosce XML. Esistono anche dei parser XML stand-alone come *xmllint command line tool*.

Principali applicazioni di XML

Le principali applicazioni di XML sono le seguenti:

- **Scambio di dati**. In presenza di informazioni provenienti da sorgenti di dati diverse (relazioni, oggetti, documenti di testo), che necessitano di essere scambiate, XML può svolgere il ruolo di linguaggio comune di interscambio.
- **Basi di dati semistrutturate**. I dati semistrutturati sono privi di uno schema regolare e non si adattano, quindi, al trattamento mediante basi di dati relazionali. XML è stato proposto quale modello dei dati per dati semistrutturati. Tale modello dei dati è molto vicino al **modello dei dati gerarchico**.

DTD (Document Type Definition)

Il markup consentito in una specifica applicazione XML può essere documentato in uno **schema**. Il linguaggio più diffuso per la definizione di schemi, l'unico presente nella specifica XML 1.0, è **DTD** (Document Type Definition). Un altro linguaggio per la definizione di schemi è **XSchema**.

Un DTD consente di imporre dei **vincoli sulla struttura** che un documento XML può assumere. Esso elenca tutti gli elementi, gli attributi e le entità (come vedremo, in XML, un'entità è un nome per una porzione di testo; alcune entità sono predefinite, altre possono essere definite dall'utente) utilizzate dal documento e specifica il contesto in cui vengono utilizzati.

I DTD non dicono **nulla** circa il tipo di **contenuto** di un elemento o il **valore** di un attributo.

DTD per il documento su Alan Turing - 1

Un DTD è precisamente una grammatica libera dal contesto per il documento.

DTD per il documento su Alan Turing:

$<!ELEMENT\ person\ (name, profession*)>$
$<!ATTLIST\ person\ born\ CDATA\ \#REQUIRED$
$\quad died\ CDATA\ \#IMPLIED>$
$<!ELEMENT\ name\ (first, last)>$
$<!ELEMENT\ first\ (\#PCDATA)>$
$<!ELEMENT\ last\ (\#PCDATA)>$
$<!ELEMENT\ profession\ (\#PCDATA)>$

La prima linea dichiara che l'elemento *person* ha esattamente un elemento figlio *name* e zero o più elementi figli *profession*, in quest'ordine.

DTD per il documento su Alan Turing - 2

La seconda linea dichiara che l'elemento *person* possiede un attributo *born* e un attributo *died* (l'ordine non è rilevante).

Le sezioni CDATA (character data) sono blocchi di character data trattati come puri dati testuali (se in una sezione CDATA sono presenti dei markup, essi non vengono riconosciuti come tali, ma trattati come character data).

La terza linea dichiara che l'elemento *name* ha due figli chiamati rispettivamente *first* e *last*, in quest'ordine.

Le ultime 3 linee specificano che gli elementi *first*, *last* e *profession* devono contenere parsed character data (PCDATA), ossia puro testo contenente entità e referenze, ma non tag.

Dichiarazione del tipo di un documento - 1

Ad un documento XML può essere associato un DTD mediante una **dichiarazione del tipo del documento**. Tale dichiarazione dovrebbe seguire la dichiarazione XML nel documento XML.

Essa ha il seguente formato:

$<!DOCTYPE\ root\ [DTD]>$

dove DOCTYPE è la keyword per la dichiarazione del tipo di documento, root è il nome dell'elemento del documento XML e DTD è l'insieme di regole che definisce il DTD (vedi lucidi precedenti).

Dichiarazione del tipo di un documento - 2

Un documento XML può contenere un **riferimento al DTD**, anziché il DTD. In questo modo lo stesso DTD può essere condiviso fra più documenti. In questo caso, la dichiarazione va riscritta nel modo seguente:

$<!DOCTYPE\ root\ KEYWORD\ "URI">$

dove KEYWORD può essere SYSTEM o PUBLIC. Nel primo caso il DTD è specificato al di fuori del documento per mezzo di un Uniform Resource Locator (URL); nel secondo caso, il DTD è specificato al di fuori del documento per mezzo di un Uniform Resource Name (URN), dove l'URN è un nome che identifica univocamente l'applicazione XML (una URL di backup può essere aggiunta per ovviare all'indisponibilità del DTD dichiarato nel file system locale).

Definizione di elementi XML in DTD - 1

Ogni elemento usato in un documento valido deve essere dichiarato nella DTD con una **definizione di elemento** della seguente forma:

$$<!ELEMENT\ name\ content>$$

dove *name* è il nome dell'elemento e *content* specifica i figli che esso deve/può avere, e in quale ordine.

La componente *content* della definizione di un elemento può essere di uno dei seguenti tipi:

- **Parsed character data**

  $$<!ELEMENT\ email\ (\#PCDATA)>$$

- **Elemento figlio**. Un elemento può contenere solo un elemento figlio di un certo tipo.

  $$<!ELEMENT\ contact\ (e\text{-}mail)>$$

Definizione di elementi XML in DTD - 2

- **Scelta**. Un elemento può contenere un tipo o un altro di figlio, ma non entrambi.

$$<!ELEMENT\ contact\ (e\text{-}mail|phone)>$$

- **Sequenza**. Un elemento può contenere più figli, in un determinato ordine.

$$<!ELEMENT\ name\ (first, last)>$$

- **Contenuto vuoto**. Un elemento non deve avere alcun contenuto, ma può avere attributi.

$$<!ELEMENT\ image\ EMPTY>$$

- **Qualsiasi contenuto**. Un elemento può avere qualsiasi contenuto (se ha dei figli, essi devono essere definiti).

$$<!ELEMENT\ image\ ANY>$$

Definizione di elementi XML in DTD - 3

- **Iterazione**. Tre diversi suffissi possono essere utilizzati per specificare quanti elementi figli vi possono essere:

  $*$ sono ammesse zero o più istanze;

  $+$ sono ammesse una o più istanze;

  ? sono ammesse zero o una istanza.

  Ad esempio, la seguente definizione impone che *name* abbia zero o più figli *first*, seguiti eventualmente da un figlio *middle* e da uno o più figli *last*.

  $$<!ELEMENT\ name\ (first*, middle?, last+)>$$

Esempi

Data la definizione precedente, tutti i seguenti elementi *name* sono validi:

$< name >$
$< first > Samuel < /first >$
$< middle > Lee < /middle >$
$< last > Jackson < /last >$
$< /name >$

$< name >$
$< first > Samuel < /first >$
$< first > Michael < /first >$
$< last > Jackson < /last >$
$< /name >$

$< name >$
$< last > Jackson < /last >$
$< last > Keaton < /last >$
$< /name >$

Altri esempi - 1

La seguente definizione di *name* consente di avere un numero arbitrario di occorrenze dei figli *first*, *middle* e *last* in qualsiasi ordine:

$$<!ELEMENT\ name\ (first|middle|last)*>$$

La seguente definizione consente di mescolare testo con markup: *name* può presentare un qualunque numero di occorrenze di figli *first*, *middle* e *last* in qualsiasi ordine, eventualmente inframezzati da testo libero.

$$<!ELEMENT\ name\ (\#PCDATA|first|middle|last)*>$$

Altri esempi - 2

Sulla base della precedente definizione, il seguente elemento *name* risulta valido:

$< name >$

$First\ comes\ the\ first\ name : < first > Samuel < /first >$

$Then\ the\ middle\ one : < middle > Lee < /middle >$

$Last\ comes\ the\ last\ name : < last > Jackson < /last >$

$Not\ very\ surprising\ indeed!$

$< /name >$

Si noti che questo è l'unico modo di caratterizzare un contenuto misto: un elemento che contiene un numero arbitrario di occorrenze di un qualunque elemento di una data lista in un qualunque ordine inframezzati da testo libero (la keyword $\#PCDATA$ deve essere la prima componente della lista).

Determinismo e non determinismo

Si consideri la seguente definizione alternativa di *name*:

$$<!ELEMENT\ name\ (first|last|(first, last))>$$

Tale definizione non è valida. Il problema è che il modello di contenuto di *name* non è deterministico.

Il problema riguarda DTD non XML: il modello di **contenuto** generato da **DTD** deve essere **deterministico**.

Quando un validatore legge un elemento XML *first*, può interpretarlo in due modi: uno è che la definizione di *name* è terminata (primo disgiunto della definizione); l'altro è che deve essere seguita da un elemento *last* (ultimo disgiunto della definizione). Leggendo lo stesso simbolo, il validatore dovrebbe poter procedere in due modi diversi (comportamento non deterministico).

Altre componenti

Un documento XML valido deve definire anche gli **attributi** degli elementi. La dichiarazione degli attributi è la seguente:

$$<!ATTLIST\ element\ attribute\ TYPE\ DEFAULT>$$

Fra gli attributi, occupano un ruolo particolare gli attributi di tipo *ID* e *IDREF*, che possono essere visti come le controparti XML delle **chiavi** e delle **chiavi esterne** delle basi di dati relazionali.

Oltre agli attributi, possono essere definite delle **entità**. Un'entità è un'abbreviazione per un certo insieme di dati, non necessariamente in formato XML. Si distingue fra diversi tipi di entità: interni o esterni, predefiniti o definiti dall'utente. Le entità interne definite dall'utente sono definite nel seguente modo:

$$<!ENTITY\ name\ "text">$$

Validità di un documento XML

Un documento XML è detto **valido** se aderisce alle condizioni del DTD ad esso associato. In generale, i browser web non validano i documenti, ma verificano solo che siano ben formati.

Se uno sta sviluppando un'applicazione, per validare il documento può usare l'API del parser. Se uno sta scrivendo il documento a mano, può usare un validatore online oppure scaricare e mandare in esecuzione un programma locale.

Un esempio di validatore online è *XML Validation Form* del Brown University Scholarly Technology Group. Un parser e validatore XML a linea di comando è *xmllint*, che fa parte della libreria XML *libxml* sviluppata per il progetto Gnome, ma utilizzabile anche al di fuori della piattaforma Gnome.

Modalità di validazione

Ad **esempio**, il documento Turing.xml può essere validato rispetto a Turing.dtd col seguente comando:

xmllint –dtdvalid Turing.dtd Turing.xml

Se Turing.xml contiene la dichiarazione del tipo di documento, è possibile utilizzare il seguente comando

xmllint –valid Turing.xml

Per evitare di produrre l'output del documento XML può essere utilizzata l'opzione –noout.

DTD e schemi relazionali - 1

I **DTD** possono essere utilizzati **come schemi**. Vi sono, però, alcune limitazioni che vanno tenute presenti:

- I sottoelementi di un elemento (interpretati come gli attributi con nome di una relazione) sono ordinati. Per svincolarsi dall'ordine, occorre elencare in modo esplicito tutti i possibili ordinamenti. Esempio: ad una relazione $R(A, B)$ deve corrispondere la dichiarazione $<!ELEMENT\ R\ ((A, B)|(B, A))>$. Lo stesso vale per l'ordine con cui gli elementi appaiono nella dichiarazione.
- Non è presente una nozione di tipo atomico. L'unico tipo atomico è PCDATA (stringa). L'assegnamento di un tipo ad un elemento o ad un attributo consentirebbe di aumentarne il contenuto semantico.
- Non è possibile specificare dei vincoli sull'insieme dei valori ammissibili per un certo elemento (ad esempio, vincolare *age* ad assumere un valore compreso tra 0 e 125).

DTD e schemi relazionali - 2

- Non è possibile vincolare il numero di occorrenze degli elementi. Ad esempio, non è possibile imporre che vi siano al più 3 occorrenze di un certo elemento.
- Il meccanismo per la gestione degli ID / IDREF è troppo semplice. Ad esempio, non è possibile restringere la condizione di unicità sugli attributi di tipo ID ad un frammento dell'intero documento. Non è possibile nemmeno vincolare il tipo degli idref (a differenza di quanto accade con le chiavi esterne del modello relazionale). Inoltre, solo singoli attributi possono essere usati come chiavi.
- Il tipo associato ad un tag di un elemento è globale. Ad esempio, in una base di dati ad oggetti un campo *name* può avere una struttura diversa a seconda della classe cui è associato. Così non è in XML. Si può in parte ovviare a tale limitazione usando tag diversi e diversi namespace.

DTD e schemi relazionali - 3

- DTD fornisce un supporto limitato per modularità, riuso ed evoluzione degli schemi. Ciò rende difficile gestire schemi di grandi dimensioni con numerose interconnessioni.
- I DTD non sono descritti usando la notazione XML (ciò avrebbe reso più agevole la loro gestione mediante strumenti XML – verifica che un DTD sia ben formato, interrogazione di schemi, ..)

In positivo, i DTD consentono di descrivere in modo immediato attributi opzionali e attributi multivalore degli schemi ER.

Esempio.

$<!ELEMENT\ R\ (A, B?, C+))>$ impone che A sia presente, B sia opzionale e C sia presente con una o più occorrenze.

Oltre i DTD: XML Schema

XML Schema è stato proposto dal W3C per risolvere tali problemi dei DTD.

In particolare, esso offre: (i) un potente sistema dei tipi, che consente di definire tipi semplici e tipi complessi, e di definire meccanismi di ereditarietà fra tipi, nello stile dei linguaggi di programmazione orientati agli oggetti, e (ii) i tipi possono essere associati sia agli elementi sia agli attributi, incrementandone il contenuto semantico.

In negativo, tali estensioni rendono XML Schema più difficile da padroneggiare, specie da parte di utenti inesperti

Linguaggi di interrogazione XML

Una **base di dati XML** è una collezione di documenti XML fra loro collegati.

Il diverso modello dei dati delle basi di dati XML (alberi) rispetto a quello delle basi di dati relazionali (tabelle) impone l'utilizzo di **linguaggi di interrogazione** differenti.

I linguaggi di interrogazione per basi di dati XML più popolari sono:

- XML Path Language (XPath). È un linguaggio che consente di recuperare elementi da un singolo documento XML.
- XML Query Language (XQuery). È un linguaggio completo per basi di dati XML (la controparte del linguaggio SQL per le basi di dati relazionali).