Il formato XML

Il testo che segue è in parte rielaborato da "Informatica Concetti e Sperimentazioni"© Apogeo 2003, in parte tratto da Wikipedia.

La persistenza dei dati

Da quando esistono i computer, i programmatori si sono scontrati con la necessità di far "vivere" i dati al di fuori dei programmi. Chiariamo il concetto con un esempio semplicissimo. Supponiamo di voler realizzare una rubrica telefonica in cui memorizzare nomi e indirizzi di amici e conoscenti. A seconda di quanto è abile il programmatore e sofisticato lo strumento, si possono fare applicazioni con un'elegante interfaccia utente e ricche di possibilità (magari capaci di comporre il numero di telefono e richiamare in continuazione se occupato). Tutte queste realizzazioni hanno un punto in comune: quando il programma non è attivo o quando il computer è spento i dati (numeri e nomi) devono restare memorizzati in modo permanente, questa necessità viene chiamata **persistenza**. Tradizionalmente, ogni programmatore decideva un formato per la rappresentazione dei dati persistenti che venivano scritti in uno o più file. Ad esempio, riferendoci alla rubrica di cui sopra, si poteva decidere di delimitare il nome con il carattere "$", oppure di riservargli un campo fisso di 30 byte o ancora di memorizzare prima il cognome e poi il nome, ecc...

La completa libertà sintattica del formato ha generato una babele di rappresentazioni, per cui un produttore di software diverso da quello originario non può fare uso dei dati altrui (a volte si tratta di migliaia di schede bibliografiche o di tutte le leggi di uno stato) se quello originario non dichiara esplicitamente e chiaramente il formato dei suoi dati. Un'altra complicazione è causata dal fatto che spesso i dati sono scritti in formati dipendenti dalla macchina, assolutamente illeggibili con programmi di uso generale o su computer di tipo diverso da quello originario.

Da qualche anno si sta affermando un standard di memorizzazione dei dati, che sembra promettere un notevole passo avanti verso una semplificazione e una razionalizzazione della persistenza.

Il linguaggio SGML (da Wikipedia)

SGML (**S**tandard **G**eneralized **M**arkup **L**anguage), è uno standard per la descrizione logica dei documenti. Discende dal Generalized Markup Language della IBM.

L'idea centrale dello standard è un tipo di marcatura generica chiamata "marcatura descrittiva" che definisce la struttura logica dei documenti.

L'organizzazione di un documento non è espressa usando la codifica dei sistemi di scrittura, che è finalizzata alla presentazione grafica, ma sono evidenziate le parti in cui è strutturato il documento (ad esempio paragrafi, capitoli) insieme ad altre particolarità del testo (come note, tabelle, intestazioni). Sono stati creati a questo scopo programmi come Alml, cioè sistemi di composizione SGML.

SGML fu inizialmente sviluppato per permettere lo scambio di documenti *machine-readable* (leggibili da un computer) in progetti governativi, legali e industriali, che devono rimanere leggibili per diverse decadi (un periodo di tempo molto lungo nell'ambito dell'informatica).

Inizialmente usato per pubblicazione di testo e basi di dati, una delle sue maggiori applicazioni fu la seconda edizione dell'Oxford English Dictionary (**OED**), che era formattato interamente usando un linguaggio SGML.

Il linguaggio HTML

HTML (**H**yper**T**ext **M**arkup **L**anguage) è un linguaggio di pubblico dominio basato su SGML la cui sintassi è stabilita dal World Wide Web Consortium (**W3C**). È stato sviluppato alla fine degli anni ottanta da Tim Berners-Lee al CERN di Ginevra. Verso il 1994 ha avuto una forte diffusione, in seguito ai primi utilizzi commerciali del web.

Nel corso degli anni, seguendo lo sviluppo di Internet, l'HTML ha subito molte revisioni, ampliamenti e miglioramenti, che sono stati indicati secondo la classica numerazione usata per descrivere le versioni dei software. Attualmente l'ultima versione disponibile è la versione 4.01, resa pubblica il 24 dicembre 1999. Nel 2007 è ricominciata l'attività di specifica con la definizione, ancora in corso, di HTML 5, attualmente allo stato di bozza. Il W3C Consortium ha annunciato che la prima versione dello standard sarà pronta per fine 2014 e l'HTML5.1 per il 2016; la prima Candidate Recommendation è stata pubblicata dal W3C il 17 Dicembre 2012.

Il linguaggio XML

Il W3C, in seguito alla guerra dei browser (ovvero la situazione verificatasi negli anni novanta nella quale Microsoft e Netscape introducevano, con ogni nuova versione del proprio browser, un'estensione proprietaria all'HTML ufficiale), fu costretto a seguire le estensioni individuali al linguaggio HTML.

Il W3C dovette scegliere quali caratteristiche standardizzare e quali lasciare fuori dalle specifiche ufficiali dell'HTML. Fu in questo contesto che iniziò a delinearsi la necessità di un linguaggio di markup che desse maggiore libertà nella definizione dei tag, pur rimanendo in uno standard.

Il **progetto XML**, che ebbe inizio alla fine degli anni ottanta nell'ambito della SGML Activity del W3C, suscitò un così forte interesse a tal punto che la W3C creò un gruppo di lavoro, chiamato XML Working Group, composto da esperti mondiali delle tecnologie SGML, ed una commissione, XML Editorial Review Board, deputata alla redazione delle specifiche del progetto.

Nel febbraio del 1998 le specifiche divennero una raccomandazione ufficiale con il nome di Extensible Mark-up Language, versione 1.0. Ben presto ci si accorse che XML non era solo limitato al contesto web, ma era qualcosa di più: uno strumento che permetteva di essere utilizzato nei più diversi contesti, dalla definizione della struttura di documenti, allo scambio delle informazioni tra sistemi diversi, dalla rappresentazione di immagini alla definizione di formati di dati.

XML (*EXtensible Markup Language*) è quindi un linguaggio di marcatura derivato dal SGML. Un documento XML è scritto in formato testo con *tag* anch'essi in formato testo, è quindi leggibile e modificabile con un comune text editor su ogni tipo di computer.

XML, come tutta la famiglia SGML, non è pensato come un linguaggio di programmazione, anche se, data la sua generalità, nulla impedirebbe di usarlo per memorizzare programmi.

Molte sono le sue analogie con HTML, ma anche profonde e significative le differenze. Innanzitutto XML non è un linguaggio orientato solamente alla visualizzazione, ma un formato assolutamente generale per descrivere dati. La visualizzazione di un documento XML è solo una delle numerose possibilità. Poiché XML, come HTML, deriva da SGML, i tag sono scritti nello stesso modo usando le parentesi angolate

`<NOMETAG>` `</NOMETAG>`

ma la sintassi è più rigida di quella dell'HTML: ogni tag aperto deve essere anche chiuso, non si possono aprire e chiudere tag in un ordine "sparso", vi è differenza tra maiuscole e minuscole. In altre parole, mentre la riga seguente è riconosciuta senza problemi dai browser HTML

```
<P><FONT size="+1"><tt> testo </font></TT>
```

per seguire la sintassi XML bisogna modificarla. As esempio, si può scrivere come segue

```
<P><FONT size="+1"><tt> testo </tt></FONT></P>
```

Con questi importanti vincoli si può dimostrare che un file XML ha una struttura ad albero.

Un'utile abbreviazione permette di sostituire un tag aperto e immediatamente chiuso

```
<BR></BR>
```

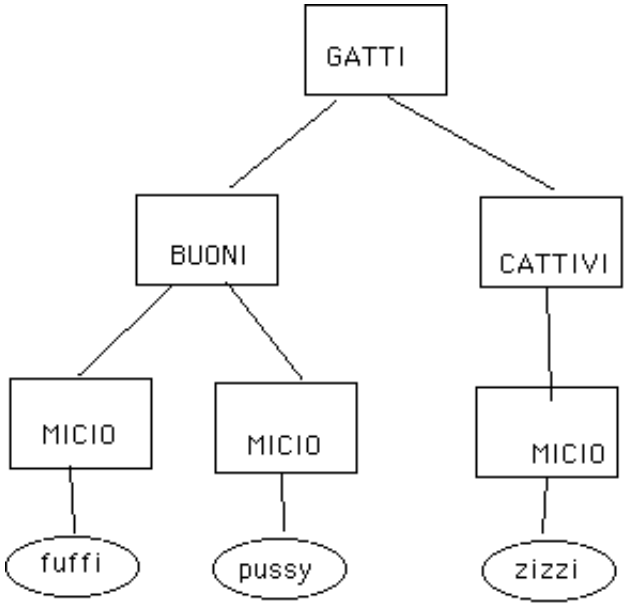
con un solo tag con la barra di chiusura in fondo

```
<BR/>
```

Ecco un primo esempio di file XML, sintatticamente corretto. Siccome i tag sono completamente liberi, ne abbiamo inventati alcuni per l'occasione.

```
<GATTI>
    <BUONI>
        <MICIO>fuffi</MICIO>
        <MICIO>pussy</MICIO>
    </BUONI>
    <CATTIVI>
        <MICIO>zizzi</MICIO>
    </CATTIVI>
</GATTI>
```

La struttura ad albero è mostrata in figura



Struttura ad albero della lista dei gatti

Una possibile interpretazione semantica è quella di un elenco di gatti buoni e cattivi; naturalmente questa è solo una deduzione psicologica indotta dai nomi. Da un punto di vista informatico il documento seguente:

```
<xyzz> <uihh> <z35>trtrt</z35> <z35>peytuut</z35>
</uihh> <hhuu> <z35>hfyryy</z35> </hhuu> </xyzz>
```

ha esattamente la **stessa** struttura, ma risulta certamente più difficile attribuirgli un significato semplicemente guardandolo.

In genere un tag XML può essere arricchito aggiungendovi degli attributi che consistono in coppie nome-valore. Anche in queste vi è completa libertà di scelta.

```
<MICIO RAZZA="soriano" PELO="lungo" VACCINATO="YES">
   fuffi
</MICIO>
```

La totale generalità della notazione XML permette di usarla per descrivere praticamente qualsiasi cosa, con il vantaggio sostanziale di una struttura sintattica univoca.

Parsing

La cosa più semplice che si può fare con un documento XML è leggerlo, farne l'analisi sintattica e utilizzarlo all'interno di un programma. Poiché le regole sintattiche dello XML sono sempre le stesse, il codice che effettua l'analisi sintattica (**parsing**) può essere scritto una volta per tutte. Nulla ovviamente impedirebbe ai programmatori di fare tutto da soli, ma le comunità XML (come **`www.xml.org`** e **`xml.apache.org`**) mettono a disposizione un insieme di strumenti già pronti.

Citiamo due pacchetti software che sono disponibili in Java, in C++ e in altri linguaggi.

- **SAX** (*Simple API for XML)*. Si tratta di un parser, che durante la lettura del file XML, ne individua i vari elementi e per ognuno di essi esegue le opportune azioni. Questo approccio si presta bene sia al trattamento di documenti molto semplici (come l'elenco dei gatti) sia per la elaborazioni di documenti molto grandi che, per la loro lunghezza, non potrebbero essere contenuti nella memoria centrale.
- **DOM** (*Document Object Module*), parser che legge l'intero documento XML e lo trasforma in un albero che risiede in memoria centrale. Questo approccio è adatto nel caso di documenti che richiedono un'elaborazione complessa (il fatto che l'albero risieda in memoria aumenta la flessibilità di programmazione), ma al tempo stesso di dimensioni tali da non creare problemi di spazio.

In Java il parser SAX è compreso nella API del linguaggio (**`javax.xml.`**... **`org.xml.sax.`**...).

E’ impossibile in questo contesto trattare in modo esauriente l’argomento. Ci limitiamo a dare alcuni esempi di codice che legge particolari file XML con il parser SAX.

Trasformazione di una lista di attributo-valore in una `HashMap`.

Esempio di dati (`file1.xml`)

```
<DATA>
<ITEM KEY="antonio"  VALUE="65774" />
<ITEM KEY="gerolamo" VALUE="57774" />
<ITEM KEY="matteo"   VALUE="78987" />
<ITEM KEY="pippo"    VALUE="45454" />
<ITEM KEY="zuzzurro" VALUE="48754" />
</DATA>
```

Ecco il codice di lettura `XMLTable` che usa il parser `SAX2` fornito dalle API di Java

```
import java.io.*;
import java.util.*;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;

public class XMLTable extends DefaultHandler {
   private HashMap<String, String> table;
   private File file;

   public XMLTable(String path) {
      this.table = new HashMap<String, String>();
      this.file = new File(path);
      if (!file.exists())
         throw new RuntimeException(" file " + path + " not found");
   }

   public Map<String, String> read() {
      try {
         SAXParserFactory.newInstance().newSAXParser().parse(file, this);
      } catch (Throwable t) {
         t.printStackTrace();
      }
      return table;
   }
```

```
/*
 * il metodo seguente ridefinisce startElement dell'interfaccia DefaultHandler e
 * viene invocato dal parser XML ogni volta che si legge un elemento
 * (ITEM nel nostro caso) name è il nome dell'elemento atts la lista degli attributi
*/
 public void startElement(String d1, String d2, String name, Attributes atts)
       throws SAXException {
    if (name.equals("ITEM")) {
       if (atts.getLength() != 2)
          throw new SAXException("wrong number of attributes: "+ atts.getLength());
       String key = null, val = null;
       for (int i = 0; i < atts.getLength(); i++) {
          String aName = atts.getQName(i);
          if (aName.equals("KEY"))   key = atts.getValue(i);
          if (aName.equals("VALUE")) val = atts.getValue(i);
       }
       if (key == null) throw new SAXException("key undefined");
       if (val == null) throw new SAXException("value undefined");
       table.put(key, val);
    }
 }
}
```

La classe XMLBatch

Vediamo il codice di una semplice classe che implementa `Iterator` e legge coppie chiave-valore da un file `XML` restituendole una per volta sotto forma di `Map`

Ecco un esempio di dati in formato XML (`file2.xml`)

```
<INPUT>
<TEST N="10" PGM="0" />
<TEST N="10" PGM="1" />
<TEST N="100" PGM="3" />
<TEST N="1000" PGM="5" />
<TEST N="10000" PGM="6" />
<TEST N="100000" PGM="7" />
<TEST N="2000000" PGM="7" />
</INPUT>
```

Ogni item di tipo `TEST` specifica una prova da eseguire dando sia il valore di *N* che il programma da eseguire, questa tecnica che è assolutamente generale permette di automatizzare in modo molto flessibile l'esecuzione ripetuta di ogni tipo di codice.

Ed ecco il codice di `XMLBatch` che usa il parser `SAX2` fornito dalle API di Java.

La scelta effettuata è stata quella di creare una classe che implementa `Iterable` e rende l'iteratore del `Vector V` interno alla classe.

```
import java.io.*;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import java.util.*;

public class XMLBatch extends DefaultHandler implements
       Iterable<Map<String, String>> {

    private Vector<Map<String, String>> V = new Vector<Map<String, String>>();
    private Map<String, String> H = new HashMap<String, String>();

    public XMLBatch(String filename) {
        File file = new File(filename);
        try {
            SAXParserFactory.newInstance().newSAXParser().parse(file, this);
        } catch (Throwable t) {
            t.printStackTrace();
        }
    }
```

```
/*
 * il metodo seguente ridefinisce startElement dell'interfaccia DefaultHandler e
 * viene invocato dal parser XML ogni volta che si legge un elemento
 * (ITEM nel nostro caso) name è il nome dell'elemento atts la lista degli attributi
 */
    public void startElement(String d1, String d2, String name, Attributes atts)
            throws SAXException {
        if (name.equals("TEST")) {
            for (int i = 0; i < atts.getLength(); i++)
                H.put(atts.getQName(i), atts.getValue(i));
            V.addElement(new HashMap<String, String>(H));
        }
    }

    public Iterator<Map<String, String>> iterator() {
        return V.iterator();
    }
}
```

Si noti che usando una `Map H` locale alla classe e mettendo nel vettore ogni volta una sua copia si lasciano invariate tutte le coppie chiave valore definite in precedenza nel file XML e non ridefinite nella riga corrente.

Organizzazione di un programma principale

Si noti che ogni test che usa come input una sequenza di coppie "chiave-valore" può essere automatizzato in questo modo senza modificare `XMLBatch` .

```
public static void main(String[] args) {
    data = new XMLBatch(filename);
    for(Map<String, String> h : data) testGenerico(h);
}
```

dove

```
void testGenerico(Map m)
```

è un metodo che effettua un'azione a partire dai dati contenuti in un elemento del file XML memorizzati in una classe che implementa `Map`.

Per chi si diverte a programmare a riga di comando (*sembra incredibile ma esistono ancora*) si può modificare il main per leggere il file di XML dalla lista degli argomenti.

```
public static void main(String[] args) {
       data = new XMLBatch(args[0]);
       for(Map<String, String> h : data) testGenerico(h);
}
```

Il programma di prova

Ecco un programma che prova le due classi.

```
import java.util.*;

public class testXML {

    public static void main(String[] args) {

        System.out.println(" test XMLTable");
        Map m = (new XMLTable("./file1.xml")).read();
        System.out.println("tabella letta");
        System.out.println(m);

        System.out.println();
        System.out.println(" test XMLBatch");

        XMLBatch data = new XMLBatch("./file2.xml");
        for(Map<String,String> H : data) testGenerico(H);
    }

    static void testGenerico(Map m) {
        System.out.println(" parametri: " + m);
    }

}
```

Ed ecco i risultati

```
test XMLTable
tabella letta
{zuzzurro=48754, pippo=45454, matteo=78987, gerolamo=57774, antonio=65774}

 test XMLBatch
 parametri: {N=10, PGM=0}
 parametri: {N=10, PGM=1}
 parametri: {N=100, PGM=3}
 parametri: {N=1000, PGM=5}
 parametri: {N=10000, PGM=6}
 parametri: {N=100000, PGM=7}
 parametri: {N=2000000, PGM=7}
```

Tecnologia XML avanzata

Linguaggi schema (permettono di creare nuovi linguaggi XML)

DTD (acronimo di Document Type Definition): è un documento attraverso cui si specificano le caratteristiche strutturali di un documento XML attraverso una serie di "regole grammaticali". In particolare definisce l'insieme degli elementi del documento XML, le relazioni gerarchiche tra gli elementi, l'ordine di apparizione nel documento XML e quali elementi e quali attributi sono opzionali o meno.

XML Schema: come la DTD, serve a definire la struttura di un documento XML. Oggi il W3C consiglia di adottarlo al posto della DTD stessa, essendo una tecnica più nuova ed avanzata. La sua sigla è XSD, acronimo di XML Schema Definition.

Altre tecnologie legate a XML

XLink: serve a collegare in modo completo due documenti XML; al contrario dei classici collegamenti ipertestuali che conosciamo in HTML, XLink permette di creare link multidirezionali e semanticamente avanzati.

XSL (acronimo di eXtensible Stylesheet Language): è il linguaggio con cui si descrive il foglio di stile di un documento XML. La sua versione estesa è l'XSLT (dove la T sta per Transformations).

XPath: è un linguaggio con cui è possibile individuare porzioni di un documento XML e sta alla base di altri strumenti per l'XML come XQuery. A supporto di questo scopo principale, fornisce

anche elementari funzionalità per trattare stringhe, numeri e dati booleani. Il suo funzionamento si basa sulla creazione di un albero a partire dal documento e la sintassi succinta permette di indirizzare una specifica parte attraverso i nodi dell'albero con la semplice parola path.

XPointer: serve ad identificare univocamente precise porzioni di un documento XML; consente poi il loro accesso ad altri linguaggi o oggetti di interfaccia.

XQuery: è un linguaggio di query concepito per essere applicabile a qualsiasi sorta di documento XML e si basa sull'utilizzo di XPath per la specificazione di percorsi all'interno di documenti. XQuery ha funzionalità che consentono di poter attingere da fonti di dati multiple per la ricerca, per filtrare i documenti o riunire i contenuti di interesse.

SVG (Scalable Vector Graphics) e **VML** (Vector Markup Language) sono standard per la creazione di immagini vettoriali che sfrutta dei documenti formattati in XML. Serve inoltre a descrivere immagini bidimensionali, statiche e dinamiche. Leggendo le istruzioni contenute nel documento sorgente XML, l'interprete disegna le figure-base fino al completamento dell'immagine.

Linguaggi XML più diffusi

XForms: come il suo nome lascia intendere, è un linguaggio nato per creare moduli (forms) di tipo HTML all'interno di un documento XML.

SMIL (Synchronized Multimedia Integration Language): questo linguaggio definito in XML, viene utilizzato per descrivere il contenuto e gestire la tempistica di presentazioni multimediali che possono combinare insieme video, audio, immagini e testo.

MathML (Mathematical Markup Language): MathML è usato per la descrizione di notazioni matematiche, procedendo a fissarne contemporaneamente struttura e contenuti, sì da poter essere riportate e processate sul Web.

X3D (eXtensible 3D): X3D è un linguaggio che premette di costruire modelli tridimensionali, siano essi semplici o sofisticati. Agli oggetti così creati possono applicarsi animazioni e meccanismi di interazione con l'utente. Come linguaggio, è costruito sul Virtual Reality Modeling Language (**VRML**), a sua volta assunto a standard internazionale nel 1997. A quest'ultimo, l'X3D acclude le capacità, tipiche dell'XML, di integrazione con le altre tecnologie del World Wide Web, di validazione dei contenuti e dell'aggiunta flessibile di nuove estensioni hardware qualora ve ne fosse necessità. In più, bisogna citare i vantaggi riguardanti la leggerezza del profilo base (Core Profile - Esistono sette profili, ognuno dei quali raggruppa un certo insieme di funzionalità comunemente utilizzate per differenti scopi. Questo permette agli sviluppatori di browser di raggiungere livelli intermedi di supporto dell'X3D, senza dover per forza implementare in una volta sola l'intera specifica) e dei browser a componenti per un download più rapido.

XML e le pagine web: XHTML

L'HTML tradizionale non è un vero standard data la sua eccessiva flessibilità ed il suo funzionamento immutato anche in presenza di errori semantici, sintattici e grammaticali. Ad affiancarlo c'è **XHTML**, ovvero l'HTML tradizionale basato su XML, con la sua struttura rigida e con le sue stesse regole.

Ad esempio in XHTML, al contrario che in HTML tradizionale, i tag vuoti vanno chiusi con uno slash (/) finale, gli attributi vuoti devono essere valorizzati con true o false, la chiusura dei tag dev'essere a specchio (se viene aperto un tag e prima di chiuderlo ne viene aperto un altro, è necessario chiudere prima il secondo tag e poi il primo), molti tag e molti attributi sono scomparsi, i caratteri speciali vanno gestiti, insieme ad altre peculiarità, ma soprattutto esiste una DTD dedicata.

L'ultima versione di XHTML è la **XHTML 1.1**.

XHTML 2 era un linguaggio di progettazione per il web, possibile successore di **XHTML1**; il 2 luglio 2009 il World Wide Web Consortium ne ha decretato la cessazione dello sviluppo, mentre era ancora in fase di specifica, a favore dell'HTML5. La definizione procedeva in parallelo a quella dell'HTML5, che veniva considerato dal World Wide Web Consortium come uno standard di markup complementare rispetto a XHTML 2, ma si è preferito abbandonare l'XHTML 2 a favore dell'HTML5, che riprende alcune delle definizioni e degli approcci concepiti per XHTML 2 ma adattati per garantire un funzionamento compatibile con gli attuali browser.