Azzolini Riccardo 2020-04-02

Document type definition

1 Linguaggio, grammatica e schema

Un insieme di documenti (o meglio, alberi) XML può essere visto come un "linguaggio".

Di solito, un *linguaggio* è un insieme di stringhe (le *parole* del linguaggio) generato / descritto da una *grammatica*. In questo caso, gli alberi XML dell'insieme considerato sono le parole del linguaggio, e, analogamente al caso di un linguaggio formato da stringhe, è possibile definire una grammatica che generi / descriva gli alberi di tale insieme. La descrizione formale di questa grammatica si chiama **schema**.

2 Linguaggi per la specifica di schemi

Rispetto, ad esempio, al modello relazionale, nel quale la specifica degli schemi è piuttosto semplice, la situazione nel caso di XML è molto più articolata. Pertanto, sono stati definiti numerosi linguaggi per la specifica di schemi XML, tra cui *DTD* e *XML Schema* (entrambi standard del W3C).

2.1 Processori di schemi

I linguaggi di schema XML vengono solitamente utilizzati nei **processori di schemi**. Un processore di schemi è un componente software che, dati in input uno schema e un documento XML, determina se il documento è **valido** rispetto allo schema (cioè se l'albero corrispondente al documento appartiene all'insieme di quelli descritti dalla grammatica specificata dallo schema).

2.2 Caratteristiche

Idealmente, un linguaggio di schema dovrebbe essere:

- *espressivo*: capace di descrivere strutture complesse, con parti "semi-strutturate" (con elementi opzionali, o che possono essere presenti in un numero variabile, ecc.);
- *efficiente*: uno schema processor deve poter svolgere efficientemente la validazione;
- *comprensibile* per gli utenti ("user-friendly").

In pratica, non si riescono sempre a conciliare tutte queste caratteristiche. Ad esempio, è necessario un compromesso tra espressività e comprensibilità, perché schemi particolarmente complessi sono difficili / impossibili da descrivere in modo facilmente comprensibile.

3 Document type definition

Come già detto, uno schema specificato tramite un **DTD** (**Document Type Definition**) definisce una grammatica che determina la "forma" (o meglio, le "forme") che gli alberi XML validi per tale grammatica possono avere.

Un DTD *non* è definito utilizzando la notazione (serializzazione) XML (a differenza di molti altri linguaggi per schemi XML). Esso è invece costituito da una sequenza di dichiarazioni di elementi, che, in pratica, stabiliscono la struttura dei nodi degli alberi XML validi secondo lo schema. Ciascuna di queste dichiarazioni ha la forma

`<!ELEMENT` *nome modello*`>`

dove:

- *nome* è il nome dell'elemento;
- *modello* specifica il contenuto dell'elemento.

Ci sono quattro tipi di modelli di contenuto:

- `EMPTY`: l'elemento non può avere figli (ma ciò non impedisce che abbia invece degli attributi).
- `ANY`: non ci sono vincoli sul contenuto, che può quindi essere una qualunque sequenza di dati carattere ed elementi.[1] Questo modello è utile soprattutto durante lo sviluppo di uno schema, per elementi che non sono ancora stati progettati.
- **Mixed content**: indica che l'elemento può contenere dati carattere "mescolati" con un numero qualsiasi di occorrenze di determinati elementi. La sintassi corrispondente a questo modello è

  `(#PCDATA |` $elemento_1$ `|` $elemento_2$ `| ... |` $elemento_n$`)*`

  dove `#PCDATA` (abbreviazione di *parsed character data*), che indica i dati carattere, è obbligatorio, ed è poi seguito da un numero qualsiasi (anche 0[2]) di (nomi di) elementi ammessi nel contenuto.

[1] Comunque, gli elementi usati all'interno di un elemento `ANY` devono essere a loro volta dichiarati nello schema; `ANY` non è un modello di "contenuto libero".

[2] Se si specifica solo `#PCDATA`, l'asterisco finale (dopo la parentesi chiusa) può essere omesso: `(#PCDATA)`.

- **Element content**: l'elemento può contenere solo altri elementi (non dati carattere), e si specificano dei vincoli sul loro ordine e numero di occorrenze, mediante il meccanismo delle *espressioni regolari*.

4 Espressioni regolari

Sia Σ un **alfabeto** (formato da un insieme di simboli/atomi). Le **espressioni regolari** su Σ sono definite in modo induttivo:

- ogni simbolo $\sigma \in \Sigma$, preso da solo, è un'espressione regolare;
- se α e β sono espressioni regolari, allora lo sono anche:

$$\begin{aligned}
\alpha? &= \text{“0 o 1 occorrenze di } \alpha\text{”} \\
\alpha* &= \text{“0 o più occorrenze di } \alpha\text{”} \\
\alpha+ &= \text{“1 o più occorrenze di } \alpha\text{”} \\
\alpha\beta &= \text{“concatenazione di } \alpha \text{ e } \beta\text{”} \\
\alpha \mid \beta &= \text{“}\alpha \text{ oppure } \beta\text{”} \\
(\alpha) &= \alpha \quad \text{(parentesi per gestire le precedenze)}
\end{aligned}$$

Tipicamente, gli operatori ?, $*$ e $+$ hanno una precedenza superiore alla concatenazione, che ha a sua volta precedenza superiore rispetto a $|$.

Un'espressione regolare denota un insieme di stringhe di simboli di Σ, costruite secondo le regole specificate da tale espressione; si dice che queste stringhe **corrispondono** all'espressione regolare. Ad esempio, dato l'alfabeto $\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$, l'espressione regolare $\mathrm{ab}* \mid \mathrm{c}$ denota l'insieme di stringhe

$$\{\mathrm{a}, \mathrm{ab}, \mathrm{abb}, \ldots, \mathrm{abbb} \cdots \mathrm{b}, \mathrm{c}\}$$

(tutte le stringhe costituite da una 'a' seguita 0 o più 'b', e la stringa formata da una singola 'c').

Le espressioni regolari formano la base di tutti i linguaggi di schema per dati semistrutturati; in questo caso, l'alfabeto Σ contiene i nomi degli elementi.

5 Esempio di dichiarazione di un elemento

Come esempio, si considera il DTD che specifica come può essere fatto un elemento `table` in XHTML. Esso contiene una sequenza di:

1. un elemento `caption`, opzionale;

2. zero o più elementi `col`, oppure zero o più elementi `colgroup`;
3. un elemento `thead`, opzionale;
4. un elemento `tfoot`, opzionale;
5. uno o più elementi `tbody`, oppure uno o più elementi `tr`.

Queste regole vengono specificate mediante la seguente dichiarazione di elemento, che contiene un'espressione regolare:[3]

```
<!ELEMENT table
    (caption?, (col* | colgroup*), thead?, tfoot?, (tbody+ | tr+))>
```

Un esempio di documento XML valido secondo questo DTD è:

```
<table>
  <caption>Città</caption>
  <thead> ... </thead>
  <tbody> ... </tbody>
</table>
```

Sarà però necessario dichiarare nel DTD anche tutti questi elementi che possono essere contenuti all'interno di `table`, specificando i possibili contenuti di ciascuno.

6 DTD e tipi

I modelli DTD sono un'approssimazione (piuttosto grezza) dei tipi dei dati che possono essere presenti all'interno di un documento XML.

In ambito XML, un tipo di dato corrisponde (circa) alla struttura di un elemento. Ad esempio, l'elemento `table` presente nel documento precedente è un'istanza del tipo `table` specificato nel DTD.

Nota: In pratica, non si parla quasi mai di "tipi" nel contesto dei DTD. Ciò è sostanzialmente per motivi "storici": l'interpretazione della struttura di un elemento come tipo di dato è più recente dell'introduzione dei DTD.

[3] A differenza della sintassi presentata prima, DTD usa una virgola per indicare la concatenazione (α, β, invece che semplicemente $\alpha\beta$).

7 Attributi

Per specificare gli attributi che un elemento può avere si usa una dichiarazione `ATTLIST` ("attribute list"):

`<!ATTLIST` *elemento definizioni*`>`

- *elemento* è il nome dell'elemento per il quale si stanno specificando gli attributi;
- *definizioni* è una lista di definizioni di attributi.

Ciascuna definizione di attributo è a sua volta formata da tre componenti:

nome tipo default

- *nome* è il nome dell'attributo.
- *tipo* specifica i possibili valori dell'attributo:
  - il tipo più comune è `CDATA` (*character data*), che ammette come valore qualunque sequenza di caratteri;
  - un altro esempio di tipo è un'*enumerazione*, che consiste nella specifica esplicita di tutti i valori (stringhe) ammessi.
- *default* determina se l'attributo sia obbligatorio o facoltativo:
  - `#REQUIRED` significa che l'attributo è obbligatorio;
  - per ottenere un attributo opzionale con default si scrive semplicemente il suo valore di default (tra virgolette);
  - `#IMPLIED` rende l'attributo opzionale, ma senza valore di default;
  - `#FIXED`, che deve essere seguito da un valore tra virgolette, indica che l'attributo è opzionale, con tale valore come default, ma se è presente allora deve per forza avere questo stesso valore.

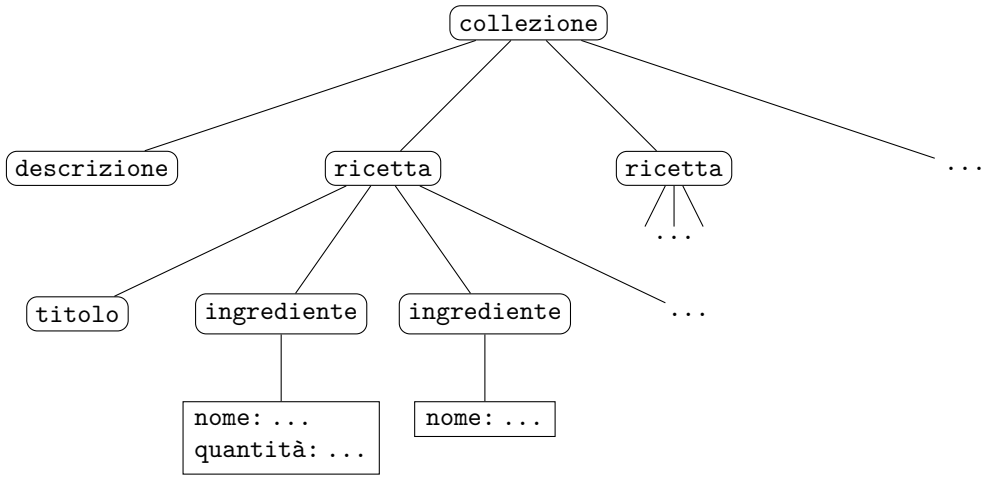
Ad esempio, la seguente dichiarazione specifica l'attributo `align` dell'elemento `p` in XHTML, che ha come tipo un'enumerazione ed è opzionale:

```
<!ATTLIST p align (left|center|right|justify) #IMPLIED>
```

8 Esempio di albero, serializzazione e DTD

Si considera, come esempio, una rappresentazione di una collezione di ricette.

- Albero astratto (corrispondente, approssimativamente, all'istanza di uno dei modelli dei dati semi-strutturati):



- Serializzazione XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<collezione>
  <descrizione> ... </descrizione>
  <ricetta>
    <titolo> ... </titolo>
    <ingrediente nome="..." quantità="..."/>
    <ingrediente nome="..."/>
  </ricetta>
  <ricetta>
    ...
  </ricetta>
  ...
</collezione>
```

  Nota: La notazione

  <*nome attributi…*/>

  è un modo compatto per scrivere un elemento vuoto: essa equivale a

  <*nome attributi…*></*nome*>

- DTD corrispondente:

```
<!ELEMENT collezione (descrizione, ricetta*)>
<!ELEMENT ricetta (titolo, ingrediente+)>
<!ELEMENT titolo (#PCDATA)>
<!ELEMENT ingrediente (#PCDATA)>
<!ATTLIST ingrediente
    nome CDATA #REQUIRED
    quantità CDATA #IMPLIED>
```

Si osserva che l'attributo `quantità` di `ingrediente` è opzionale, e che gli elementi `collezione` e `ricetta` non specificano il numero preciso di elementi figli che devono avere: i dati sono semi-strutturati.

9 Limitazioni di DTD

DTD ha molte importanti limitazioni, che lo rendono poco espressivo. Alcune delle principali sono:

- Non sono disponibili tipi di dati diversi dalle stringhe di caratteri. Ad esempio, non si può specificare che un elemento o un attributo deve contenere un valore numerico.
- Non è possibile imporre alcun tipo di vincolo ai dati carattere.
- Non c'è modo di specificare che un elemento è presente solo quando sono presenti/assenti altri elementi (o attributi); in generale, non si possono definire strutture che dipendono dal contesto.
- I namespace non sono supportati.

Per risolvere (la maggior parte di) questi problemi, è stato definito il linguaggio XML Schema.