XML Schema

Fabio Vitali

Sommario

Oggi esaminiamo in breve XML Schema

- Perché non bastano i DTD
- Tipi ed elementi
- Definizione di elementi ed attributi
- Altri aspetti rilevanti di XML Schema

Motivazione (1)

Inizialmente si pensava che XML servisse solo per i documenti

XML è più semplice di SGML, è più generale ed aperto di HTML, è lo strumento ideale per esprimere documenti di testo, siano essi libri, manuali o pagine Web.

Quindi l'enfasi iniziale era su internazionalizzazione, strutturazione, facilità di conversione, ecc. Raccoglieva in pieno l'eredità di SGML. Lo sviluppo di XML era difatti condotto da membri della comunità SGML.

Motivazione (2)

Nasce poi l'idea che XML possa servire per qualcosa di più: XML è (anche) un linguaggio di markup per trasferire dati: un meccanismo per convertire dati dal formato interno dell'applicazione ad un formato di trasporto, facile da convertire in altri formati interni.

Non pensato per la visione umana, ma per essere prodotto ed usato da programmi.

XML è un'interfaccia (Adam Bosworth):

- Un'interfaccia tra autore e lettore, attraverso XSL e XLink, per portare significato tra creatore ed utente
- Un'interfaccia tra applicazione ed applicazione, attraverso XML Schema, per esprimere contratti sui formati, e verificarne il rispetto.

A seguire: Motivazione (3)

Motivazione (3)

Tutta la faccenda del trasferimento dei dati si semplifica: i documenti strutturati e gerarchici sono un formato ragionevole di sintassi praticamente per qualunque cosa: documenti di testo, record di database, ecc.

Nella W3C Note di Agosto 1999 (`http://www.w3.org/TR/schema-arch`)
"Many data-oriented applications are being defined which build their own data structures on top of an XML document layer, effectively using XML documents as a transfer mechanism for structured data; ”

Validazione e buona forma

La buona forma di un documento XML è una proprietà puramente sintattica.

La validazione, viceversa, è la verifica di un impegno preso **sopra** al formato, ad un livello già semantico:

- *Mi impegno a scrivere dei documenti che siano formati da capitoli, ciascuno con un titolo e vari paragrafi, e ogni immagine con la sua didascalia.*

Per esprimere documenti di testo, i DTD probabilmente bastano, ma per esprimere blocchi di dati strutturati, è necessario un meccanismo di verifica più raffinato.

XML Schema è stato pensato per fornire quel supporto di validazione che i DTD permettono solo parzialmente, in particolare sul contenuto degli elementi e degli attributi del documenti XML.

XML Schema e DTD (1)

Modularità

- I DTD offrono le entità parametriche per tutto: content model ripetuti, frammenti di DTD riusabili, modularizzazione delle specifiche.
- XML Schema offre:
  - Un meccanismo di inclusionie di file(differenziato e sofisticato)
  - Un sistema di tipi gerarchico e complesso
  - Un sistea di specifica di frammenti riutilizzabili di content model e attributi.

Gestione dei namespace

- I DTD ignorano i namespace. E' molto complesso far convivere DTD e qualifica degli elementi.
- XML Schema permette esplicitamente di qualificare gli elementi e gli attributi, e fornisce molte funzionalità di estensione o limitazione dei vincoli sugli elementi a seconda del loro namespace.

A seguire: XML Schema e DTD (2)

XML Schema e DTD (2)

Vincoli su elementi e attributi

- I DTD permettono un ragionevole controllo degli elementi strutturati, ma poca flessibilità sui content model misti.
- Inoltre non permettono vincoli sugli elementi di testo (#PCDATA e CDATA) a parte le liste di valori negli attributi
- XML Schema amplia il set di vincoli per gli elementi strutturati, ed è molto più flessibile negli elementi con CM misto.
- Inoltre permette di definire tipi semplici, ovvero di esprimere vincoli su dati di tipo stringa, in maniera completa e sofisticata.

Documentazione esplicita

- Allegare documentazione per esseri umani in un DTD significa inserire commenti XML dentro al DTD. Questo è limitante e fragile (i parser eliminano i commenti).
- XML Schema permette di inserire annotazioni in maniera esplicita e controllata, in maniera che sopravviva al parser.

XML Schema e DTD (3)

Sintassi XML

- I DTD usano una sintassi propria e particolare, che richiede parser appositi e strumenti di generazione e verifica appositi.
- XML Schema usa XML come sintassi. Tutti gli strumenti che si usano per XML sono immediatamente utili: parser, visualizzatori, verificatori, ecc.
- Per contro, XML Schema è estremamente più verboso, tre o quattro volte più lungo del corrispondente DTD, e molto spesso meno chiaro da leggere.
- Ci sono state molte obiezioni su questo specifico aspetto, ma TBL ha specificamente insistito per ricondurre tutto ad XML (altrimenti, che meta-sintassi sarebbe?)

XML Schema

E’ una delle attività del working group su XML. Ha prodotto 7 generazioni di working draft, per poi divenire recommendation nel maggio del 2001.

E’ diviso in tre parti:

- XML Schema Part 0: Primer (un’introduzione)
- XML Schema Part 1: Structures (struttura del documento XML Schema)
- XML Schema Part 2: Datatypes (modello dei dati e meccanismi di estensione dei tipi)

Formato di un XML Schema

Un documento di XML Schema è racchiuso in un elemento <schema>, e può contenere, in varia forma ed ordine, i seguenti elementi:

- **`<import>`** ed **`<include>`** per inserire, in varia forma, altri frammenti di schema da altri documenti
- **`<simpleType>`** e **`<complexType>`** per la definizione di tipi denominati usabili in seguito
- **`<element>`** ed **`<attribute>`** per la definizione di elementi ed attributi **globali** del documento.
- **`<attributeGroup>`** e **`<group>`** per definire serie di attributi e gruppi di content model complessi e denominati.
- **`<notation>`** per definire notazioni non XML all'interno di un documento XML
- **`<annotation>`** per esprimere commenti per esseri umani o per applicazioni diverse dal parser di XML Schema.

I tipi in XML Schema

XML Schema usa i tipi per esprimere vincoli sul contenuto di elementi ed attributi.

- Un tipo semplice è un tipo di dati che non può contenere markup e non può avere attributi. In pratica è una sequenza di caratteri. E' una specificazione (e restrizione) di CDATA o #PCDATA.
- Un tipo complesso è un tipo di dati che può contenere markup e avere attributi. E' l'equivalente di un tipo strutturato o misto.

XML predefinisce un grande numero di tipi semplici: string, decimal, float, boolean, uriReference, date, time, ecc.

Ogni tipo semplice è caratterizzato da alcune proprietà, dette *facets*, che ne descrivono vincoli e formati (permessi ed obblighi).

E’ data possibilità di derivare nuovi tipi, sia per restrizione che per estensione di permessi ed obblighi.

Tipi semplici

Gli elementi e gli attributi sono istanze di un tipo. I tipi semplici sono tipi stringa non ulteriormente strutturati, e possono essere usati per entrambi.

XML Schema non fa nessuna distinzione tra attributi ed elementi con content model testo.

- `<xsd:element name="price" type="xsd:decimal"/>`
- `<xsd:attribute name="code" type="xsd:ID"/>`

Sono predefiniti molti tipi semplici, che possono essere usati liberamente nelle definizioni. Il nome di un tipo semplice predefinito appartiene allo stesso namespace di XML Schema.

E' possibile derivare i tipi semplici per restrizione, unione o lista.

Una lista parziale di tipi semplici

- **string**: una stringa di caratteri.
- **boolean**: i valori 'true ' e 'false'.
- **decimal**: una stringa di numeri (con segno e punto): '-34.15 '
- **float**: un reale in notazione scientifica: '12.78E-12 '.
- **duration** : una stringa per una durata temporale nel formato PnYnMnDTnHnMnS. Ad esempio 'P1Y2M3DT10H30M ' è la durata di 1 anno, 2 mesi, 3 giorni, 10 ore, e 30 minuti.
- **date** : una data nel formato CCYY-MM-DD: '2001-04-25 '.
- **time** : un valore di orario nel formato hh:mm:ss con una appendice opzionale per l'indicazione del fuso orario. Es.: '13:20:00+01:00 ' significa 1:20 PM in Middle European Time (+01:00).
- **hexBinary**: dati binari arbitrari in formato esadecimale: '0FB7'.
- **anyURI**: la stringa di un URI, come "http://www.w3.org/ ". Accetta sia URI relativi che assoluti.
- **ID , IDREF**: Una stringa senza whitespace con le stesse proprietà e vincoli di ID e IDREF nei DTD.

Derivazione per restrizione

Si parte da un tipo già definito e ne si restringe il set di valori leciti attraverso l'uso di facet:

```
<xsd:element name="editore" type="Teditore"/>
<xsd:simpleType name="Teditore">
      <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Addison Wesley"/>
            <xsd:enumeration value="Academic Press"/>
            <xsd:enumeration value="Morgan Kaufmann"/>
      </xsd:restriction>
</xsd:simpleType>
<xsd:element name="data" type="Tdatarecente"/>
<xsd:simpleType name=" Tdatarecente">
      <xsd:restriction base="xsd:date">
            <xsd:minInclusive value="2002-01-01"/>
      </xsd:restriction>
</xsd:simpleType>
```

Facets

Per ogni tipo io posso precisare dei facets, delle caratteristiche indipendenti tra di loro che specificano aspetti del tipo:

- length, minLength, maxLength: numero richiesto, minimo e massimo di caratteri
- minExclusive, minInclusive, maxExclusive, maxInclusive: valore massimo e minimo, inclusivo ed esclusivo
- precision, scale: numero di cifre significative e di decimali significativi
- pattern: espressione regolare che il valore deve soddisfare
- enumeration: lista all'interno dei quali scegliere il valore (simile alla lista di valori leciti degli attributi nei DTD).
- period, duration, encoding, ecc.

Derivazione per unione

L'insieme dei valori leciti è data dall'unione dei valori leciti di due tipi semplici.

```
<xsd:element name="prezzo" type="Tprezzo">
<xsd:simpleType name="Tprezzo">
  <xsd:union>
    <xsd:simpleType>
      <xsd:restriction base="xsd:decimal">
        <xsd:minExclusive value="0.0"/>
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="gratis"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>
```

Derivazione per lista

Sono valori leciti una lista separata da virgole di valori del tipo semplice specificato.

```
<xsd:simpleType name="TListaDiNumeri">
  <xsd:list itemType='xsd:decimal'/>
</xsd:simpleType>
<xsd:attribute name="coord" type="TListaDiNumeri"/>


<area coord="25,30,75,90"/>
```

Tipi anonimi e tipi denominati

In XML Schema i tipi possono essere predefiniti (solo x tipi semplici), denominati (con una definizione esplicita, come nei casi precedenti) o anonimi (interni alla definizione di un elemento)

```
<xsd:element name="editore">
   <xsd:simpleType>
      <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Addison Wesley"/>
            <xsd:enumeration value="Academic Press"/>
            <xsd:enumeration value="Morgan Kaufmann"/>
      </xsd:restriction>
   </xsd:simpleType>
</xsd:element>
```

Tipi complessi

Sono tipi complessi:

- Content model ANY ed EMPTY
- Content model strutturati
- Content model misti
- Content model di **qualunque tipo** ma con attributi

Non esistono tipi complessi predefiniti (o quasi).

La derivazione può avvenire per restrizione o estensione.

Content model ANY ed EMPTY

```
<xsd:element name="anything" type="xsd:anyType"/>

<xsd:element name="empty">
       <xsd:complexType/>
</xsd:element>

<xsd:complexType name="emptyTypeWithAttributes">
       <xsd:attribute name="first"  type="xsd:string"/>
       <xsd:attribute name="second" type="xsd:decimal"/>
</xsd:complexType>
<xsd:element name="empty2" type="emptyTypeWithAttributes"/>
```

Content model strutturati (1)

Come nei DTD si usano virgole e caret per specificare obblighi e scelte tra gli elementi di un content model complesso, così in XML schema si usano <choice>, <sequence> e <all>. Questi sostituiscono anche le parentesi.

- La sequenza (A, B, C) diventa

```
<xsd:sequence>
   <xsd:element name=”A” type="xsd:string"/>
   <xsd:element name=”B” type="xsd:string"/>
   <xsd:element name=”C” type="xsd:string"/>
</xsd:sequence>
```

- La scelta (A | B | C) diventa

```
<xsd:choice>
   <xsd:element name=”A” type="xsd:string"/>
   <xsd:element name=”B” type="xsd:string"/>
   <xsd:element name=”C” type="xsd:string"/>
</xsd:choice>
```

A seguire: Content model complessi (2)

Content model complessi (2)

- XML Schema riprende l'operatore & di SGML: tutti gli elementi debbono essere presenti, ma in qualunque ordine. (A & B & C) diventa:

```
<xsd:all>
  <xsd:element name="A" type="xsd:string"/>
  <xsd:element name="B" type="xsd:string"/>
  <xsd:element name="C" type="xsd:string"/>
</xsd:all>
```

  N.B.: ci sono restrizioni: la struttura all può solo essere l'unica struttura di un tipo complesso (non posso usarla in espressioni più complesse).

- Il raggruppamento non ha bisogno di parentesi: (A, (B | C)) diventa

```
<xsd:sequence>
  <xsd:element name="A" type="xsd:string"/>
  <xsd:choice>
    <xsd:element name="B" type="xsd:string"/>
    <xsd:element name="C" type="xsd:string"/>
  </xsd:choice>
</xsd:sequence>
```

A seguire: Content model complessi (3)

Content model complessi (3)

Per specificare ripetibilità e facoltatività, si usano gli attributi minOccurs e maxOccurs:

XML Schema permette non solo i valori 0, 1 e infinito, ma qualunque numero intero. Infinito è "unbounded", e può essere usato solo per maxOccurs. Per default entrambi valgono 1. Inoltre, minOccurs ≤ maxOccurs

```
<xsd:element name="optional" type="x" minOccurs="0"/>
<xsd:element name="repeat" type="x" maxOccurs="unbounded"/>
<xsd:element name="free" type="x"
             minOccurs="0" maxOccurs="unbounded"/>
<xsd:element name="counted" type="x"
             minOccurs="2" maxOccurs="4"/>
```

Posso specificare questi attributi non solo per gli elementi, ma anche per le strutture (choice, sequence, all). Ad esempio, (A,B,C)* diventa:

```
<xsd:sequence minOccurs="0" maxOccurs="unbounded">
   <xsd:element name="A" type="xsd:string"/>
   <xsd:element name="B" type="xsd:string"/>
   <xsd:element name="C" type="xsd:string"/>
</xsd:sequence>
```

A seguire: Content model misti (1)

Content model misti (1)

Il content model misto aggiunge semplicemente l'attributo mixed con valore "true".

Qualunque espressione di elementi viene rispettata, ma il PCDATA può comparire ovunque, prima o dopo questi elementi.

```
<!ELEMENT para   (#PCDATA | em | strong)*>
```

```
<xsd:complexType name="paraContent" mixed="true">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="em" type="paraContent"/>
    <xsd:element name="strong" type="paraContent"/>
  </xsd:choice>
</xsd:complexType>
<xsd:element name="para" type="paraContent"/>
```

A seguire: Content model misti (2)

Content model misti (2)

Ma si possono fare cose più complesse:

```
<xsd:element name="review">
  <xsd:complexType mixed="true">
    <xsd:sequence>
      <xsd:element name="author" type="xsd:string"/>
      <xsd:element name="title" type="xsd:string"/>
      <xsd:element name="pub" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

In questo caso il #PCDATA può comparire ovunque prima e dopo questi elementi, ma questi debbono essere posti esattamente in quell'ordine e in quel numero.

Content model con attributi

Qualunque elemento preveda attributi è necessariamente di un tipo complesso. XML Schema differenzia infatti tra tipi complessi con contenuto semplice e tipi complessi con contenuto complesso.

Questa è la definizione di un tipo il cui contenuto è semplice ma che prevede un attributo.

```
<xsd:complexType name="prezzo">
  <xsd:simpleContent>
    <xsd:extension base="xsd:decimal">
      <xsd:attribute name="valuta" type="xsd:string"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
<xsd:element name="price" type="prezzo"/>
```

Derivare tipi complessi

Ci sono due modi per derivare tipi complessi:

- **Per restrizione**: si limitano ulteriormente i vincoli espressi: modificando minOccurs e maxOccurs, fissando dei valori per certi elementi o attributi, o imponendo ad un elemento un sottotipo del tipo originario.
- **Per estensione**: aggiungendo al content model nuovi elementi o nuovi attributi. Attenzione: i nuovi elementi sono posti necessariamente alla fine degli altri.

Derivazione per restrizione

```
<xsd:complexType name='nomecognome'>
   <xsd:sequence>
      <xsd:element name='nome' type='xsd:string'
           minOccurs='0' maxOccurs='unbounded'/>
      <xsd:element name='cognome' type='xsd:string'/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name='cognome' >
   <xsd:restriction base="nomecognome">
     <xsd:element name='cognome' type='xsd:string'/>
   </xsd:restriction>
</xsd:complexType>

<xsd:complexType name='nome' >
   <xsd:restriction base="nomecognome">
     <xsd:element name='nome' type='xsd:string'/>
   </xsd:restriction>
</xsd:complexType>
```

Perché questa no?

A seguire: Derivazione per estensione

Derivazione per estensione

```
<xsd:complexType name='nomecognome'>
   <xsd:sequence>
      <xsd:element name='nome' type='xsd:string'
          minOccurs='0' maxOccurs='unbounded'/>
      <xsd:element name='cognome' type='xsd:string'/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name='nomecontitolo' >
   <xsd:extension base="nomecognome">
      <xsd:sequence>
         <xsd:element ref='title' minOccurs='0'/>
      </xsd:sequence>
   </xsd:extension>
</xsd:complexType>
```

Definire elementi ed attributi

- Si usano gli elementi <element> e <attribute>.
- Se sono posti all'interno del tag schema sono elementi ed attributi *globali* (possono essere root di documenti).
- Altrimenti sono usabili solo all'interno di elementi che li prevedono come tipo.
- Questi hanno vari attributi importanti:
  - Name: il nome del tag o dell'attributo (*definizione locale*)
  - Ref: il nome di un elemento o attributo definito altrove (*definizione globale*)
  - Type: il nome del tipo, se non esplicitato come content
  - maxOccurs, minOccurs: il numero minimo e massimo di occorrenze
  - Fixed, default, nullable, ecc.: specificano valori fissi, di default e determinano la possibilità di elementi nulli.

A seguire: Definizioni locali o globali (1)

Definizioni locali o globali (1)

Una definizione si dice **globale** se è posta all'interno del tag <schema>. In questo caso l'elemento o l'attributo è definito in maniera assoluta. L'elemento può essere un elemento radice del documento.

Una definizione si dice **locale** se è inserita all'interno di un tag <complexType>. In questo caso l'elemento o l'attributo esiste solo se esiste un'istanza di quel tipo, e l'elemento non può essere un elemento radice del documento.

A seguire: Definizioni locali o globali (2)

Definizioni locali o globali (2)

E' possibile all'interno di un tipo complesso fare riferimento ad un elemento globale, usando l'attributo ref invece che name:

```
<xsd:schema... >
  <xsd:element name="librieriviste">
    <xsd:complexType>
      <xsd:choice maxOccurs="unbounded">
       <xsd:element ref="libro"/>
       <xsd:element ref="rivista"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="libro" type="xsd:string"/>
  <xsd:element name="rivista" type="xsd:string"/>
</xsd:schema>
```

Gruppi e gruppi di attributi

E’ possibile raccogliere gli elementi e gli attributi in gruppi:

```
<xsd:element name=”A”>
  <xsd:complexType>
    <xsd:group ref=”elemA” />
    <xsd:attributeGroup ref=”attrsA” />
  </xsd:complexType>
</xsd:element>

<xsd:group name=“elemA” />
<xsd:sequence>
      <xsd:element name=”B” type="xsd:string"/>
      <xsd:element name=”C” type="xsd:string"/>
    </xsd:sequence>
</xsd:group>

<xsd:attributeGroup name=”attrsA">
  <xsd:attribute name=”p” type=”xsd:string”/>
  <xsd:attribute name=”q” type=”xsd:string”/>
</xsd:attributeGroup>
```

Annotazioni

Nei DTD l'unico posto dove mettere note e istruzioni di compilazione sono i commenti. Però i commenti sono a perdere: possono essere mangiati in qualunque fase dell'elaborazione.

In XML Schema, invece, esiste un posto specifico dove mettere note ed istruzioni, l'elemento <annotation>.

L'elemento <annotation> può contenere elementi <documentation> (pensati per essere letti da esseri umani) oppure elementi <appInfo>, pensati per essere digeriti da applicazioni specifiche

```
<xsd:element name='pippo'>
  <annotation>
    <documentation>elemento pippo</documentation>
  </annotation>
  ... Il resto della definizione
</xsd:element>
```

I namespace (1)

La dichiarazione di targetNamespace definisce il namespace del documento da validare.

Gli attributi elementFormDefault e attributeFormDefault permettono di controllare se l'uso del prefisso è necessario per i tipi non globali.

```
<schema xmlns="http://www.w3.org/2000/08/XMLSchema"
        xmlns:po="http://www.example.com/PO1"
        targetNamespace="http://www.example.com/PO1"
        elementFormDefault="unqualified"
        attributeFormDefault="unqualified">
   <element name="A"   type="po:prova"/>
   <element name="C"   type="string"/>
   <complexType name="po:prova">
     <sequence>
         <element name="B" type="string" >
         <element ref="C" />
     </sequence>
   </complexType>
</schema>
```

I namespace (2)

Quello che i namespace permettono di fare è di specificare regole di validazione solo su alcuni e non tutti i namespace del documento:

```
<element name="htmlElement">
   <complexType>
      <sequence>
         <any namespace="http://www.w3.org/1999/xhtml"
              minOccurs="1" maxOccurs="unbounded"/>
      </sequence>
   </complexType>
</element>
```

Nell'attributo namespace dell'elemento <any> posso specificare o un namespace vero e proprio, o i valori:

- ##any: qualunque XML ben formato
- ##local: qualunque XML non sia qualificato (cioè privo di dichiarazione di namespace)
- ##other: qualunque XML tranne il target namespace

Local e global scope (1)

Non funziona!

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:b="http://www.betterbooks.org"
      targetNamespace="http://www.betterbooks.org" >
  <xsd:element name="book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="author" type="xsd:string"/>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="pub" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
<b:book xmlns:b="http://www.betterbooks.org">
   <b:author>Douglas Adams</b:author>
   <b:title>Hitch-hikers Guide to the Galaxy</b:title>
   <b:pub>Pan Books</b:pub>
</b:book>
```

A seguire: Local e global scope (2)

Local e global scope (2)

Neanche questo funziona!

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.betterbooks.org" >
  <xsd:element name="book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="author" type="xsd:string"/>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="pub" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
</xsd:schema>

<book xmlns="http://www.betterbooks.org">
      <author>Douglas Adams</author>
      <title>Hitch-hikers Guide to the Galaxy</title>
      <pub>Pan Books</pub>
</book>
```

A seguire: Local e global scope (3)

Local e global scope (3)

Questo invece funziona!

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.betterbooks.org" >
  <xsd:element name="book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="author" type="xsd:string"/>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="pub" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
</xsd:schema>

<b:book xmlns:b="http://www.betterbooks.org">
      <author>Douglas Adams</author>
      <title>Hitch-hikers Guide to the Galaxy</title>
      <pub>Pan Books</pub>
</b:book>
```

A seguire: Local e global scope (4)

Local e global scope (4)

Gli attributi elementFormDefault e attributeFormDefault controllano se gli elementi e gli attributi locali siano per default qualificati o non qualificati.

Per default sono NON qualificati, il che è ragionevole per gli attributi, e un po' una sorpresa per gli elementi. Si noti ad esempio che in RDF sia gli elementi che gli attributi debbono essere qualificati.

Per rendere la o la seconda versione del documento valido, dobbiamo porre elementFormDefault a qualificato, altrimenti non funziona niente.

A seguire: Local e global scope (5)

Local e global scope (5)

Così funziona!

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.betterbooks.org"
      elementFormDefault="qualified">
  <xsd:element name="book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="author" type="xsd:string"/>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="pub" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

<b:book xmlns:b="http://www.betterbooks.org">
   <b:author>Douglas Adams</b:author>
   <b:title>Hitch-hikers Guide to the Galaxy</b:title>
   <b:pub>Pan Books</b:pub>
</b:book>
```

Inclusioni e importazioni

In XML Schema, esistono meccanismi per dividere lo schema in più file, o per importare definizioni appartenenti ad altri namespace

- Include: Le nuove definizioni appartengono allo stesso namespace, ed è come se venissero inserite direttamente nel documento.
- Redefine: come include, le definizioni appartengono allo stesso namespace, ma possono venire ridefiniti tipi, elementi, gruppi, ecc.
- Import: le nuove definizioni appartengono ad un altro namespace, ed è l'unico modo per fare schemi che riguardino namespace multipli.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.w1.org"
            xmlns="http://www.w1.org"
            xmlns:cd="http://www.w2.org">
  <import namespace="http://www.w2.org"
          schemaLocation="http://www.w2.org/s1.xsd"/>
```

Unicità e chiavi

In XML Schema, è possibile richiedere che certi valori siano unici, o che certi valori siano chiavi di riferimenti, analoghi alla coppia ID/IDREF in XML “classico”. Tuttavia, è possibile specificare pattern molto complessi come elementi chiave.

- Unicità:

```
<xsd:unique name="code">
      <xsd:selector xpath="/books/book"/>
      <xsd:field xpath="author"/>
      <xsd:field xpath="title"/>
</xsd:unique>
```

- Integrità referenziale

```
<xsd:key name="code">
      <xsd:selector xpath="/books/book"/>
      <xsd:field xpath="@code"/>
</xsd:key>
<xsd:keyref name="list" refer="code">
      <xsd:selector xpath="/list/objects"/>
      <xsd:field xpath="@id"/>
</xsd:keyref>
```

Riferirsi ad uno schema

```
<fv:pippo xmlns:fv ="http://www.fabio.org/Pippo"
        xmlns:xsi="http://www.w3.org/2000/08/XMLSchema"
        xsi:schemaLocation="http://www.fabio.org/pippo.xsi">
…
</fv:pippo>
```

Con l'attributo schemaLocation dentro all'istanza del documento XML diamo un suggerimento sulla posizione dello schema al validatore (ma la stessa informazione può essere data off-line, ad esempio perché predefinita, o in un header della connessione HTTP).

Conclusioni

Oggi abbiamo parlato di XML Schema:

- Motivazioni e status
- Organizzazione dei tipi
- Definizione di elementi ed attributi
- Content model, gruppi ed altri aspetti

Riferimenti

- P. V. Biron, A. Malhotra,*XML Schema Part 2: Datatypes*, W3C Recommendation 02 May 2001, http://www.w3.org/TR/xmlschema-2/
- D. C. Fallside, *XML Schema Part 0: Primer*, W3C Recommendation, 2 May 2001,http://www.w3.org/TR/xmlschema-0/
- A. Malhotra, M. Maloney, *XML Schema Requirements*, W3C Note 15 February 1999, http://www.w3.org/TR/NOTE-xml-schema-req
- H. S. Thompson, D. Beech, M. Maloney, N. Mendelsohn, *XML Schema Part 1: Structures*, W3C Recommendation 2 May 2001,http://www.w3.org/TR/xmlschema-1/