XSD

XML Schema Definition

Luca Bartoli

Sezione 1

Aaaaaaa

aaaaaaaaaaaaaaa

aaaaaaaaaaaaaaaaah!!

L'astronave ShipML è in orbita, controllata da un complesso sistema robotizzato, che riceve in input da varie basi sulla terra file di istruzioni, che (per motivi ignoti) sono in formato XML. Ogni errore, anche piccolo, potrebbe essere fatale, e l'astronave potrebbe esplodere o spazzare via i malcapitati terrestri!

Un esempio di file potrebbe essere:

```
<?xml version="1.0" encoding="utf-8"?>
<commands>
   <auth>
      <user degree="5"> Luis Pollo </user>
      <base zone="B32"> Seattle </base>
      <password>Password12</password>
   </auth>
   <engine>
      <run combust="hydrogen"> 5 </run>
   </engine>
   <move left="1" up="4"/>
```

```
  <shoot weapon="laser">
      <target>
         <galaxy>Andromeda</galaxy>
         <planet>KELT-2Ab</planet>
         <coordinates>
            <E>012-29-32</E>
            <N>41-53-24</N>
         </coordinates>
      </target>
      <target>
         <galaxy>Milky Way</galaxy>
         <planet>Mars</planet>
         <coordinates>
            <N>12-4-12</N>
            <E>90-3-16</E>
         </coordinates>
      </target>
   </shoot>
</commands>
```

Ben Formato

Si ricorda che un documento XML ben formato ha le seguenti caratteristiche:

- Inizia con l'intestazione XML
- Ha un elemento radice
- A tutti i tag aperti, esclusi quelli vuoti, corrisponde il tag di chiusura
- Gli elementi sono *case sensitive*
- Tutti gli elementi devono essere innestati correttamente.
- Tutti i valori degli attributi devono essere racchiusi da virgolette
- I caratteri speciali vanno riferiti tramite entità

Generalità

In molte applicazioni risulta sufficiente che il documento sia ben formato, in altri frangenti invece occorre una validazione (come nel caso della nostra astronave!):

- Gli XML schema language sono linguaggi utilizzati per descrivere o porre vincoli sul contenuto di documenti XML
- L' **X**ML **S**chema **D**efinition è uno schema language rilasciato dal W3C
- Il linguaggio, basato a sua volta sull'XML, riproduce ed estende considerabilmente le capacità del DTD.
- Ad oggi nel panorama dell'XML la maggior parte dei formati è descritta in XSD, in quanto risulta facile da imparare ed è dotato di una vasta capacità espressiva.

Lo scopo del Xml Schema Definition è quello di definire:

- Gli elementi e attributi che possono apparire in un documento
- Il numero (e l'ordine) dei nodi figli
- Tipi (semplici e complessi) per gli elementi e attributi
- Valori di default e costanti per elementi e attributi

Alcuni punti di forza del linguaggio sono:

- si può utilizzare lo stesso schema per una classe di documenti
- si possono creare i propri tipi di dato derivati
- è possibile specificare più schemi sullo stesso documento e dividere lo stesso schema in più file

Note storiche

- Pubblicato dal World Wide Web Consortium nel 2001.
- Unico *schema language* a raggiungere la validazione ufficiale del consorzio.
- Chiamato sia **WXS** che **XSD**, ma il W3C ha adottato il secondo come nome ufficiale.
- Prende chiaramente molto dal DTD e incorpora molte delle *features* presenti negli vari *schema language*, costituendo un buon compromesso.

Sezione 2

Intestazione

- Uno schema è di per se un documento XML, perciò deve inziare con:

```
<?xml version="1.0" encoding="utf-8"?>
```

- Il *root node*, che completa l'intestazione, e che racchiude tutto il contenuto dello schema è:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="https://www.spaceship.com"
  xmlns="https://www.spaceship.com"
  elementFormDefault="qualified">
  ...
</xs:schema>
```

- I vari attributi sono:
  - xmlns:xs indica che gli elementi e i tipi utilizzati nello schema sono definiti nel *namespace* specificato e che vanni preceduti dal prefisso xs
  - targetNamespace indica che gli elementi utilizzati nello schema sono definiti nel *namespace* specificato (attributo opzionale)
  - xmlns imposta il namespace di default (quando un tag non è preceduto da alcun prefisso si sottintende xmlns:)
  - elementFormDefault se impostato a "qualified" indica che tutti i nomi presenti nel documento devono essere preceduti da un *namespace*.

Elementi e Attributi

- Per dichiarare un elemento si scrive:

```
<xs:element name="<name>" type="<type>"/>
```

- Analogamente, gli attributi si dichiarano con:

```
<xs:attribute name="<name>" type="<type>"/>
```

- Per specificare valori di default si utilizza l'attributo default
- Per impostare un valore costante per un elemento si utilizza l'attributo fixed
- Per rendere un attributo obbligatoria sia aggiunge use = "required"

Tipi

- l'attributo type può assumere i valori, corrispondenti a tipi primitivi forniti dall'XSD:
  - xs:string
  - xs:decimal
  - xs:integer
  - xs:boolean
  - xs:date
  - xs:time

Esempio

```
  ...
<xs:element name="user" type="xs:string"/>
...
<xs:attribute name="degree" type="xs:integer" default="0" use="
    required"/>
...
```

Tipi Semplici

- Per specificare un nuovo tipo semplice per un elemento, che ha come base un tipo primitivo e in più dei vincoli di dominio, si può scrivere:

```
<xs:element name="<name>">
   <xs:simpleType>
    <xs:restriction base="<primitive-type>">
      ...
    </xs:restriction>
   </xs:simpleType>
</xs:element>
```

- Se si vuole utilizzare lo stesso tipo per più elementi ed attributi si può definire una sola volta assegnandogli un nome:

```
<xs:simpleType name="<type-name>">
  <xs:restriction base="<primitive-type>">
    ...
  </xs:restriction>
</xs:simpleType>

<xs:element name="<element-name1>" type="<type-name>" />
<xs:element name="<element-name2>" type="<type-name>" />
```

Restrizioni

- Se base="xs:integer" allora al posto dei tre puntini si possono porre vincoli con:

```
<xs:minInclusive value="<min-value>"/>
<xs:maxInclusive value="<max-value>"/>
```

  Ne esistono anche le varianti esclusive (maxExclusive, minExclusive)

- Per restringere il dominio a pochi valori predefiniti (come per gli *enum*) allora al posto dei tre puntini si può utilizzare un elenco:

```
<xs:enumeration value="<value1>"/>
<xs:enumeration value="<value2"/>
...
```

- Se base="xs:decimal" allora al posto dei tre puntini,per specificare cifre totali e cifre dopo la virgola si può utilizzare:

```
<xsd:totalDigits value="<total-digits>"/>
<xsd:fractionDigits value="<fraction-digits>"/>
```

- Se base="xs:string" allora al posto dei tre puntini si possono specificare dei pattern sui quali testare la stringa, del tutto analoghi alle *RegExp* in vari linguaggi di programmazione (Java, Javascript, Python...):

```
<xs:pattern value="<pattern-to-match>"/>
```

  per vincolare la sola lunghezza:

```
<xs:length value="<length>"/>
```

  In alternativa esiste anche maxLength e minLength

Esempio String

```
<xs:simpleType name="pswtype">
 <xs:restriction base="xs:string">
   <xs:pattern value="[a-zA-Z0-9]*[A-Z]+[a-zA-Z0-9]*[0-9]+[a-
        zA-Z0-9]*"/>
   <xs:minLength value="8"/>
 </xs:restriction>
</xs:simpleType>
```

Esempio Enum

```
<xs:attribute name="weapon">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:enumeration value="laser" />
   <xs:enumeration value="nuke" />
   <xs:enumeration value="catapult" />
  </xs:restriction>
 </xs:simpleType>
</xs:attribute>
```

Tipi Complessi

- Analogamente ai tipi semplici, per dichiarare un elemento complesso si può utilizzare il costrutto nel seguente modo:

```
<xs:element name="<name>">
  <xs:complexType>
    ...
  </xs:complexType>
</xs:element>
```

- In alternativa per creare un tipo complesso ed attribuirgli un nome, per poi usarlo come tipo nella dichiarazione di nuovi elementi:

```
<xs:complexType name="<complex-type>">
  ...
</xs:complexType>
...
<xs:element name="<name3>" type="<complex-type>"/>
```

- Per il tag simpleType abbiamo mostrato l'utilizzo delle restrizioni. Con i tipi complessi la sintassi diventa:

```
<xs:complexType name="<derived-type>">
  <xs:complexContent>
    <xs:restriction base="<base-type>">
      ...
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

- Si possono inoltre estendere i tipi con una sintassi analoga:

```
<xs:complexType name="<derived-type>">
  <xs:complexContent>
    <xs:extension base="<base-type>">
      ...
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Elementi Complessi

Distinguiamo quattro classi di elementi complessi:

- Elementi vuoti
- Elementi che contengono solo altri elementi
- Elementi che contengono solo testo
- Elementi che contengono sia testo che altri elementi

Elementi Vuoti

- Per dichiarare elementi vuoti si deve specificare un tipo complesso, e dove andrebbero dichiarati il tipo o gli elementi contenuti non si specifica nulla.

- E’ auspicabile che un elemento vuoto abbia quanto meno un attributo.

Esempio Vuoto

```
<xs:simpleType name="motion">
  <xs:restriction base="xs:integer">
    <xs:minExclusive value="0"/>
    <xs:maxInclusive value="100"/>
  </xs:restriction>
</xs:simpleType>
<xs:element name="move">
  <xs:complexType>
    <xs:attribute name="right" type="motion"/>
    <xs:attribute name="left" type="motion"/>
    <xs:attribute name="up" type="motion"/>
    <xs:attribute name="down" type="motion"/>
  </xs:complexType>
</xs:element>
```

Solo Elementi

- In questo caso non si fa altro che specificare gli elementi contenuti
- Per l'ordine e il numero di occorrenze si usano gli indicatori, come sequence il cui significato verrà chiarito nel seguito.

Esempio Solo Elementi

```
<xs:element name="auth">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="user" type="xs:string" />
      <xs:element name="base">
      ...
      </xs:element>
      <xs:element name="password" type="pswdtype" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Solo Testo

- Il titolo **Solo Testo** può trarre in inganno: il contenuto di questo tipo di elementi non è solo di tipo *xs:string*, si intende solo che non contiene altri tag.

- Risulta necessario quando si ha un elemento che contiene solo testo, ma che ha uno o più attributi.

- La sintassi è quella dell'esempio che segue, in generale vanno utilizzati extension e restriction, per rispettivamente, aggiungere attributi o vincolare il dominio dell'elemento.

Esempio Solo Testo

```
<xs:simpleType name="engineRange">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="1" />
    <xs:maxInclusive value="10" />
  </xs:restriction>
</xs:simpleType>
...
<xs:element name="run">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="engineRange">
        <xs:attribute name="combust" type="xs:string" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Testo ed Elementi

- Un esempio potrebbe essere:

```
<msg>
  Dear <to>rldnjn</to>,

  dksdfb ksdbfkjsbdf kjsbkjsb kjbk
  sdfsdfa sdfasdf asdfasdfasdf asdffsa

  Best regards,
  <from>alnfasn</from>
</msg>
```

  Per spedire messaggi alieni dall'astronave.

- In tal caso la sintassi è identica a quella del caso con solo elementi, ma con l'aggiunta del valore booleano mixed impostato a vero.

Esempio Testo ed Elementi

```
<xs:element name="msg">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="to" type="xs:string" />
      <xs:element name="from" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Indicatori

Esistono tre gruppi di indicatori:

- Quelli che controllano l'ordine degli elementi:
  - all
  - choice
  - sequence
- Quelli che controllano il numero di occorrenze:
  - minOccurs
  - maxOccurs
- Quelli che gestiscono il raggruppamento:
  - group
  - attributeGroup

Ordine

- all specifica che gli elementi possono apparire in qualsiasi ordine.

Esempio All

```
<xs:element name="coordinates">
  <xs:complexType>
    <xs:all>
      <xs:element name="E" type="coords">
      <xs:element name="N" type="coords">
    </xs:all>
  </xs:complexType>
</xs:element>
```

- choice specifica che può comparire solo uno degli elementi figli:

Esempio Choice

```
<xs:element name="engine">
  <xs:complexType>
    <xs:choice>
      <xs:element name="shut" type="engineRange" />
      <xs:element name="run" type="engineRange">
        ...
      </xs:element>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

- sequence specifica l'ordine in cui devono occorrere gli elementi:

Esempio Sequence

```
<xs:element name="target">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="galaxy" type="xs:string" />
      <xs:element name="planet" type="xs:string" />
      <xs:element name="coordinates">
      ...
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Occorrenze

- minOccurs serve per impostare il minimo numero di occorrenze di un elemento, il valore di default è 1
- maxOccurs serve per impostare il massimo numero di occorrenze di un elemento, di default è unbounded

Esempio occorrenze

```
<xs:element name="shoot">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="target" maxOccurs="10">
      ...
      </xs:element>
    </xs:sequence>
    <xs:attribute name="weapon">
    ...
    </xs:attribute>
  </xs:complexType>
</xs:element>
```

Raggruppamento

- group serve per specificare o riferire gruppi di elementi

```
<xs:group name="<group-name>">
  <xs:all>
    <xs:element name="<name1>" type="<type1>"/>
    ...
  </xs:all>
</xs:group>
```

- attributeGroup serve per specificare o riferire gruppi di attributi

```
<xs:attributegGroup name="<attr-group-name>">
  <xs:attribute name="<attr-name1>" type="<attr-type1>"/>
  ...
</xs:attributeGroup>
```

Riferimenti

Invece di dichiarare direttamente *inline* elementi ed attributi si possono descrivere separatamente, e poi si può applicare un riferimento:

- Supponendo di aver dichiarato l'elemento "<element-name>", si può scrivere ad esempio:

```
<xs:element name="<some-name>">
  <xs:complexType>
    ...
    <xs:element ref="<element-name>" minOccurs="2"/>
    ...
  </complexType>
</xs:element>
```

- Supponendo di aver dichiarato l'attributo "<attr-name>", si può scrivere:

```
<xs:element name="<some-name>">
  <xs:complexType>
    ...
    <xs:attribute ref="<attr-name>" use="required"/>
    ...
  </complexType>
</xs:element>
```

- Supponendo di aver dichiarato il gruppo "<group-name>", si può scrivere:

```
<xs:element name="<some-name>">
  <xs:complexType>
    ...
    <xs:group ref="<group-name>" minOccurs="0"/>
    ...
  </complexType>
</xs:element>
```

- Supponendo di aver dichiarato il gruppo di attributi "<attr-group-name>", si può scrivere:

```
<xs:element name="<some-name>">
  <xs:complexType>
    ...
    <xs:attributeGroup ref="<attr-group-name>" use="
        required"/>
    ...
  </complexType>
</xs:element>
```

Any

- Il tag any serve per dare la possibilità di inserire un qualsiasi tag, vincolando l'ordine rispetto agli altri elementi, e specificarne il numero di occorrenze.

```
<xs:sequence>
  <xs:element name="<name1>" type="<type1">/>
  <xs:element name="<name2>" type="<type2>"/>
  <xs:any minOccurs="0"/>
  ...
 </xs:sequence>
```

  Il tag inserito può poi essere vincolato da altri componenti specificati nello stesso file o in altri.

- Analogamente, anyAttribute serve per specificare la possibilità di inserire un qualsiasi attributo su un tag.

```
<xs:complexType>
  <xs:sequence>
     ...
  </xs:sequence>
  <xs:anyAttribute/>
 </xs:complexType>
```

Specificare gli Schemi

Per specificare gli schemi che un documento utilizza si utilizza la seguente riga di intestazione, da inserire sul *root node*:

```
<commands
xmlns="https://www.spaceship.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="URL-TO-SCHEMA/commands.xsd ...">
```

Per specificare più schemi basta scriverli come valore dell'attributo schema location, al posto dei tre puntini.

GRAZIE!

La nostra eroica validazione ha protetto la terra dall'essere spazzata via, grazie!