Codifica di Testi - XML Schema e Intro TEI a.a. 2018-2019

Angelo Mario Del Grosso

angelo.delgrosso@ilc.cnr.it

CNR-ILC-LicoLab

Istituto di Linguistica Computazionale "A. Zampolli",
12th October 2018

Contenuto della lezione

Progress status

Cos'è uno schema XML

Uno schema XML è un documento XML standard che descrive come deve essere realizzato un altro documento XML.
Ci riferiamo a questa tecnologia con l'acronimo XSD.

A cosa serve uno Schema XML

I documenti XSD sono usati per validare documenti XML.
Tuttavia un documento XSD viene realizzato tramite l'uso di un vocabolario predefinito riferibile attraverso un namespace con URI standard.

XSD Schema

Il termine XSD o XML Schema denota un documento XML che descrive e valida la struttura e il contenuto di un altro documento XML.

XSD Schema

Dichiarazione del documento (declaration) e istanza del documento (instance).

XSD elemento root

L'elemento radice di uno schema XSD è sempre l'elemento "`<schema>`".
Tutte le definizione devono seguire quindi l'elemento "`<schema>`".

XSD Schema

Tutti gli elementi e gli attributi dello schema sono dichiarati all'interno del namespace "`http://www.w3.org/2001/XMLSchema.`".
Tutti i documenti XSD contengono la dichiarazione a questo namespace con prefisso convenzionale **xsd** oppoure **xs**.

XSD componenti di base

I componenti di base di uno Schema XSD sono le dichiarazioni degli elementi e le dichiarazioni degli attributi.

XSD Schema

Le dichiarazioni più complesse si poggiano su queste unità: elementi e attributi.

XSD dichiarazioni

Scrivere un pezzo di codice XSD per descrivere e validare un elemento per un documento XML è detto *element declaration*.

XSD dichiarazioni di base

XSD permette di dichiarare elementi, attributi e di specificare il numero di figli, le occorrenze, l'ordine di apparizione, e i tipi di dati del content model.

Element Types: simple and complex

La dichiarazione di un elemento può avere un tipo semplice (*simple type*) oppure un tipo complesso (*complex type*) a seconda della sua struttura e del suo contenuto.

Simple Type e Complex Type

La dichiarazione di un elemento ha un tipo semplice se non possiede **né figli né attributi**.
La dichiarazione di un elemento ha un tipo complesso in tutti gli altri casi.

XSD esempio

```
<xsd:schema
xmlns:xsd=‘‘http://www.w3.org/2001/XMLSchema’’>
<xsd:element name=‘‘text’’/>
</xsd:schema>
```

XSD esempio elemento di tipo semplice

```
<text>Il primo documento XML Validato</text>
```

XML XSD esempio

Il documento XML istanza dello schema XSD per essere valido deve contenere un elemento radice. Validare il documento XML con il relativo XSD con XMLlint.

XMLlint

```
xmllint xmlfirst.xml --schema
../schema/xsd/xsdfirst.xsd
```

Element Complex Types: esempio

```
<xsd:schema
xmlns:xsd=‘‘http://www.w3.org/2001/XMLSchema’’>
<xsd:element name=‘‘Employee’’> <xsd:complexType>
<xsd:attribute name=‘‘FirstName’’/>
</xsd:complexType> </xsd:element> </xsd:schema>
```

Element Complex Types: esempio

Il documento XML istanza dello schema:

```
<Employee FirstName="Jacob"/>
```

Complex Types

Alla base dello standard XSD ci sono le dichiarazioni degli elementi e degli attributi, ad un livello di astrazione più alto ci sono i types e i groups.
A complex type can have attributes, child elements or both.
Here is another example that shows a complex type having child elements.

Element Complex Types: Esempio Elemento con Figlio

```
<xsd:schema
xmlns:xsd=‘‘http://www.w3.org/2001/XMLSchema’’>
<xsd:element name=‘‘text’’> <xsd:complexType>
<xsd:sequence> <xsd:element name=‘‘body’’/>
```

Espressività dell'XSD

- Attribute Group, Element Group
- Order Indicators: all, sequence, choice
- Occurrence Indicators: minOccurs and maxOccurrs
- Annotation (utili per documentare le dichiarazioni)

Espressività dell'XSD

- Data types: Built-in
- FACETS per una validazione oculata dei valori (elemento o un attributo).

Element Declaration

An XSD element declaration represents an element in the XML instance document An element is said to have a Simple Type if it does not have any attributes and does not have child elements.

Element Declaration: Istanza XML

```
<p>il contentuto testuale di un paragrafo</p>
```

Global and Local element declaration

The advantage of using a global element declaration is that it can be reused (referred) at other locations from within the same schema.
When you have a global element declaration, you can refer it in multiple locations in your schema (level of reusability).

Global element declaration: esempio

```
<xsd:schema
xmlns:xsd=‘‘http://www.w3.org/2001/XMLSchema">
<xsd:element name=‘‘body’’> <xsd:complexType>
<xsd:attribute name=‘‘lang’’/> <xsd:attribute
name=‘‘type’’/> </xsd:complexType> </xsd:element>
```

Group declaration

Commonly used attributes and elements can be grouped together into Attribute Groups and Element Groups. You can then refer to such a group at multiple locations in your schema definition. Attribute groups and Element groups provide a certain level of reusability.

Global element declaration: esempio

```
<xsd:group name=‘‘fileDesc’’> <xsd:sequence>
<xsd:element name=‘‘titleStmt’’/> <xsd:element
name=‘‘publicationStmt’’/> <xsd:element
name=‘‘sourceDesc’’/> </xsd:sequence>
```

Global element declaration: esempio

```
<xsd:group ref=‘‘fileDesc’’/>
```

Element Declaration: Attributi

Attributi della dichiarazione di elementi: The only mandatory attribute that an element declaration should take is the "name" attribute.

Element Declaration: Attributi - Esempio

```
<xsd:element name=''TEI''/>
```

Element declaration: lista Attributi

- **name** (*g-l*)

Element declaration: lista Attributi (cont.)

- **final** (g) limits

Element declaration: Attributi - Esempio

```
<xsd:element name=‘‘body’’> <xsd:complexType>
<xsd:sequence maxOccurs=‘‘unbounded’’>
<xsd:element ref=‘‘div’’/> </xsd:sequence>
</xsd:complexType> </xsd:element>
```

Element declaration: Attributi - Esempio

```
<xsd:element name=‘‘div’’ type=‘‘divType’’ />
<xsd:complexType name=‘‘divType’’> [...]
</xsd:complexType>
```

Attribute declaration:

An attribute is declared with ¡xsd:attribute¿ element. The only mandatory attribute of an attribute declaration is “name”. When an attribute is declared right under the “¡xsd:schema¿” element, it is called global attribute declaration. When it is declared within a Complex Type, it is called Local attribute declaration.

Attribute declaration

```
 <xsd:attribute name=‘‘Name’’
type=‘‘xsd:string’’/>
```

Qualsiasi dichiarazione di attributo per essere effettivamente

Attribute declaration: Attributi

- **name** (*g-I*) the name of the attribute as it should appear in the XML instance (Mandatory)
- id (g-I) used by the schema processor to uniquely identify the XSD components within a given schema
- type (g-I) associates a data type with an attribute (facilitates validation on the value). make sure that only valid values are accepted

Attribute declaration: Attributi (cont.)

- **default** (*g-l*) The default attribute assigns a default value to an attribute declaration (solo se l'attributo non è presente)
- fixed (g-l) It prevents the attribute from taking any value other than the pre-defined one.
- ref (l) A globally declared attribute can be inserted into a complex type by using the "ref" attribute.
- use (l) specifies whether the attribute is optional or mandatory (values optional or required, prohibited).
- form (l) specifies whether the attribute needs to be qualified by a namespace prefix or not, in the XML

Attribute declaration: Esempio (global)

```
 <xsd:schema
xmlns:xsd=‘‘http://www.w3.org/2001/XMLSchema’’>
<xsd:attribute name=‘‘analysis’’ /> <!-- -->
<xsd:element name=‘‘word’’> <xsd:complexType>
<xsd:attribute ref=‘‘analysis’’/>
</xsd:complexType> </xsd:element> </xsd:schema>
```

Global attribute declarations are useful when an attribute is declared with several validations

Attribute declaration: Attribute Groups

Attribute Groups provide a convenient means to reuse attribute declarations in multiple complex types. Attribute Groups provide a better level of reusability by grouping one or more attribute declarations into a named group. By using an attribute group, you can avoid this repetition of code. Chain of attribute group hierarchies.

Attribute declaration: Esempio (global)

```
 <xsd:schema
xmlns:xsd=‘‘http://www.w3.org/2001/XMLSchema’’>
<xsd:attributeGroup name=‘‘EmpAttributes’’> <!--
Declaration of attribute ‘‘name’’ -->
<xsd:attribute name=‘‘name’’> <xsd:simpleType>
<xsd:restriction base=‘‘xsd:string’’>
<xsd:maxLength value=‘‘20’’/> </xsd:restriction>
</xsd:simpleType> </xsd:attribute> <!--
Declaration of attribute ‘‘department’’ -->
<xsd:attribute name=‘‘department’’>
<xsd:simpleType> <xsd:restriction
base=‘‘xsd:string’’> <xsd:length value=‘‘2’’/>
</xsd:restriction> </xsd:simpleType>
```

Attribute declaration: Esempio (global)

```
 [...]  <!-- Declaration of root element
‘‘Employees’’ --> <xsd:element name=‘‘Employees’’>
<xsd:complexType> <xsd:sequence> <!-- Declaration
of ‘‘Manager’’ element --> <xsd:element
name=‘‘Manager’’> <xsd:complexType>
<xsd:attributeGroup ref=‘‘EmpAttributes’’/>
</xsd:complexType> </xsd:element> <!-- Declaration
of ‘‘department’’ element --> <xsd:element
name=‘‘TechLead’’> <xsd:complexType>
<xsd:attributeGroup ref=‘‘EmpAttributes’’/>
</xsd:complexType> </xsd:element> </xsd:sequence>
</xsd:complexType> </xsd:element> </xsd:schema>
```

XSD Data Type

Programming languages use data types to make sure that correct values are stored to variables and correct operations are done using those variables.

XSD Data Type

When you associate a variable to a data type you are basically restricting the values that the variable can store and restricting the permissible operations on them.

XSD Data Type: esempio - dichiarazione

```
<xsd:schema
xmlns:xsd=‘‘http://www.w3.org/2001/XMLSchema’’>
<xsd:element name=‘‘div’’> <xsd:complexType>
<xsd:attribute name=‘‘type’’ /> <xsd:attribute
name=‘‘n’’ type=‘‘xsd:integer’’/>
</xsd:complexType> </xsd:element> </xsd:schema>
```

XSD Data Type: esempio - istanza

```
<div type="chapter" n="1" />’ (corretto)
<div type="chapter" n="uno" />’ (errato)
```

XSD Data Type

XSD supports a number of different data types to describe and validate almost all values that we might need to work with. It supports deriving new data types from the built-in data types

XSD Data Type

Data types help describe a certain piece of data more accurately and help validate them more efficiently.
XSD supports almost fifty data types. They can be divided into Primitive Data Types, Derived Data Types

Primitive Data Types

Primitive Data Types are base data types from which other data types are derived.
XSD has nineteen primitive data types

Primiteive Data Types

string	boolean	decimal
float	double	duration
dateTime	time	date
gYearMonth	gYear	gMonthDay
gDay	gMonth	hexBinary
base64Binary	anyURI	QName
NOTATION		

Primitive Data Types

Primitive Data Types are the base data types of XSD. This means that they themselves have not been derived from another type.

Derived Data Types

These are Data Types derived directly or indirectly from Primitive Data Types.

Elementi per la definizione degli schemi xml
principi XSD

Derived Type	Base Type
normalizedString	string
Integer	decimal

Derived Type	Base Type
Token	normalizedString
NMTOKEN	token
Name	token
Language	token
NMTOKENS	NMTOKEN
NCName	Name
ENTITY	NCName
ENTITIES	ENTITY

Elementi per la definizione degli schemi xml
principi XSD

Derived Type	Base Type
nonPositiveInteger	integer
nonNegativeInteger	integer
Long	integer
negativeInteger	nonPositiveInteger
int	Long
short	int
byte	short
unsignedLong	nonNegativeInteger
positiveInteger	nonNegativeInteger
unsignedInt	unsignedLong
unsignedShort	unsignedInt
unsignedByte	unsignedShort

Data Type: Facets

Each data type has a number of properties that can be restricted to perform additional validations on the value. These properties are called **Facets** in XSD.

Data Type: Facets

So each data type has a certain number of predefined Facets. A facet controls a certain attribute or characteristic of a data type

Data Type: Facets Esempio

```
 <xsd:schema
xmlns:xsd=‘‘http://www.w3.org/2001/XMLSchema’’>
<xsd:element name=‘‘name’’> <xsd:complexType>
<xsd:attribute name=‘‘type’’> <xsd:simpleType>
<xsd:restriction base=‘‘xsd:string’’> <xsd:length
value=‘‘15’’/> </xsd:restriction>
</xsd:simpleType> </xsd:attribute>
</xsd:complexType> </xsd:element> </xsd:schema>
```

Data Type: Facets Esempio

```
 <xsd:schema
xmlns:xsd=‘‘http://www.w3.org/2001/XMLSchema’’>
<xsd:element name=‘‘name’’> <xsd:complexType>
<xsd:attribute name=‘‘type’’> <xsd:simpleType>
<xsd:restriction base=‘‘xsd:string’’> <xsd:patter
value=‘‘[A-Za-z]+’’/> </xsd:restriction>
</xsd:simpleType> </xsd:attribute>
</xsd:complexType> </xsd:element> </xsd:schema>
```

Elementi per la definizione degli schemi xml
principi XSD

	pattern	whitespace	length	minLength	maxLength	enumeration	totalDigits	fractionDigits	minInclusive	maxInclusive	minExclusive	maxExclusive
string	✓	✓	✓	✓	✓	✓						
boolean	✓	✓										
decimal	✓	✓				✓	✓	✓	✓	✓	✓	✓
float	✓	✓				✓			✓	✓	✓	✓
double	✓	✓				✓			✓	✓	✓	✓
duration	✓	✓				✓			✓	✓	✓	✓
dateTime	✓	✓				✓			✓	✓	✓	✓
time	✓	✓				✓			✓	✓	✓	✓
date	✓	✓				✓			✓	✓	✓	✓
gYear	✓	✓				✓			✓	✓	✓	✓
gMonthDay	✓	✓				✓			✓	✓	✓	✓
gDay	✓	✓				✓			✓	✓	✓	✓
gMonth	✓	✓				✓			✓	✓	✓	✓
haxBinary	✓	✓	✓	✓	✓	✓						
base64 Binary	✓	✓	✓	✓	✓	✓						
anyURI	✓	✓	✓	✓	✓	✓						
QName	✓	✓	✓	✓	✓	✓						

Simple Type vs Complex Type

The basic distinction between simple types and complex types is that only a complex type can contain child elements and attributes.

Simple Type vs Complex Type

Simple types can only store a value. An element or attribute can have a simple type.

Elementi per la definizione degli schemi xml
principi XSD


```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="ZipCode" type="zipType" />
  <xsd:simpleType name="zipType">
    <xsd:restriction base="xsd:integer">
      <xsd:maxInclusive value="99999"/>
      <xsd:minInclusive value="10000"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="ZipCode">
    <xsd:simpleType>
      <xsd:restriction base="xsd:integer">
        <xsd:maxInclusive value="99999"/>
        <xsd:minInclusive value="10000"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:element>
</xsd:schema>
```

Simple Types can be declared globally or locally

Global simple types

Global simple types help reuse the definitions as well as help organize and maintain the schema.

helpful when the same set of validations is to be performed

simple types example

```
<xsd:simpleType name=‘‘chapterNumberType’’>
<xsd:restriction base=‘‘xsd:integer’’>
<xsd:maxInclusive value=‘‘1000’’/>
<xsd:minInclusive value=‘‘1’’/> </xsd:restriction>
</xsd:simpleType>
```

simple types example

```
<xsd:element name=‘‘item’’> <xsd:complexType>
<xsd:attribute name=‘‘originalChapter’’
type=‘‘chapterNumberType’’/> </xsd:complexType>
</xsd:element>
```

simple types: deriving

Un nuovo tipo può essere derivato da un tipo già dichiarato (primitivo o meno) ed ereditarne le caratteristiche.

simple types example

- Derive by restriction
- Derive by list
- Derive by Union

simple types: deriving

This involves identifying a base type that is close to what we are looking for, and adding the additional restrictions or validation rules.

simple types: deriving by Restriction

A restriction is defined by adding "xsd:restriction" to the Simple Type declaration. Each data type has a number of properties that restricts the set of values it can accept (facets). When you derive a new data type by restriction, you restrict one or more facets.

simple types: deriving by Restriction

```
<xsd:simpleType name=‘‘signatureType’’>
<xsd:restriction base=‘‘xsd:integer’’>
<xsd:totalDigits value=‘‘5’’/> </xsd:restriction>
</xsd:simpleType>
```

simple types: deriving by Restriction

```
<xsd:element name=‘‘signature’’
type=‘‘signatureType’’>
<signature>12345</sgnature>  (valido)
<signature>123ab</signature>  (non valido)
```

simple types: deriving by List

data type can store a SPACE separated list of values accepted by the base type

simple types: deriving by List

```
<xsd:simpleType name=‘‘chapterNumberList’’>
<xsd:list itemType=‘‘xsd:integer’’ />
</xsd:simpleType>
 <xsd:element name=‘‘chapters’’
type=‘‘chapterNumberList’’ />
 <chapters>1 53 60 61 205 409</chapters>
```

simple types: deriving by Union

The derived type can store the values acceptable to any of the base types from which the new type is derived.

simple types: deriving by Union

```
<xsd:simpleType name=‘‘ZipCityUnion’’> <xsd:union>
<xsd:simpleType> <xsd:restriction base="ZipType"/>
</xsd:simpleType> <xsd:simpleType>
<xsd:restriction base="CityType"/>
</xsd:simpleType> </xsd:union> </xsd:simpleType>
```

simple types: deriving

The value will be accepted only if it validates successfully with one of the base types.

simple types: deriving

It is not allowed to make the value space of a derived type less restrictive than the base type.

simple types: deriving facets

each XSD data type has a certain number of facets that control its value space.
When we derive a new Simple Type from another, the new type will inherit all the facets of the base type.

simple types: deriving facets

You can set the **fixed** attribute of the given facets to **true** to make sure that the derived types do not modify those facets.

Simple types: controllare la derivazione

XSD provides a way to protect your Simple Type so that no other Types can inherit from it.

controllare la derivazione: l'attributo final

- restriction
- list
- union
- extension
- #all

Simple types: controllare la derivazione - esempio

```
 <xsd:schema
xmlns:xsd=‘‘http://www.w3.org/2001/XMLSchema’’>
<xsd:simpleType name=‘‘zipType’’
final=‘‘restriction union list extension’’>
<xsd:restriction base=‘‘xsd:integer’’>
<xsd:maxInclusive value=‘‘99999’’ fixed=‘‘true’’/>
<xsd:minInclusive value=‘‘10000’’/>
</xsd:restriction> </xsd:simpleType> </xsd:schema>
```

Simple types: controllare la derivazione - esempio

Extension refers to deriving a new type from a Simple Type that results in a Complex Type

Simple types: controllare la derivazione - esempio

When the **final** attribute is set to **#all**, the Simple Type cannot be inherited at all.

Progress status

Progress status