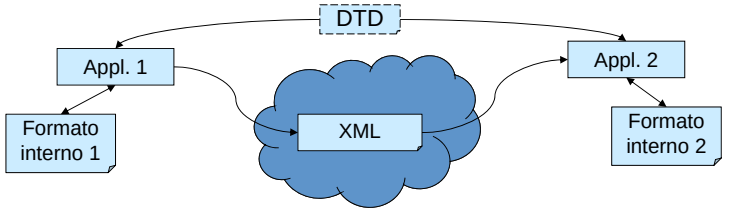
XML Schema

Ombretta Gaggi
Università di Padova

XML per lo scambio dei dati

- XML = un linguaggio di markup per trasferire dati dal formato interno dell'applicazione ad un formato di trasporto
  - interoperabilità dei dati (facilità di conversione, estrazione, ecc.)



Perché XML Schema?

- Per esprimere documenti di testo, i DTD probabilmente bastano, ma non bastano per esprimere blocchi di dati strutturato: è necessario un meccanismo di verifica più raffinato
- XML Schema è un linguaggio XML
- DTD è poco pratico e poco espressivo per le esigenze di comunicazione
- DTD non permette di specificare vincoli di integrità referenziale
- Modularità: i DTD offrono le entità parametriche per tutto
  - XML Schema offre:
    - un meccanismo di inclusioni di file(differenziato e sofisticato)
    - un sistema di tipi gerarchico e complesso
    - un sistema di specifica di frammenti riutilizzabili di content model e attributi

Perché XML Schema?

- Namespace: è difficile far coesistere DTD e namespace perché i primi sono nati con XML, mentre i namespace sono stati introdotti successivamente
- 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

Limiti dei DTD

- 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. Non esiste il concetto di testo "tipizzato"
  - es: <libro> <titolo>Codice Da Vinci</titolo>
    <prezzo> 30 </prezzo></libro>
    ..ma anche…
    <prezzo> trenta </prezzo>

    <!ELEMENT libro (titolo,prezzo) >
    <!ELEMENT titolo (#PCDATA)>
    <!ELEMENT prezzo (#PCDATA)>
  - <corso titolo="Tecnologie Web">
    <numeroIscritti>Marco</numeroIscritti>
    </corso>

I DTD NON sono scritti in 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.
- Svantaggi di XMLSchema:
  - Per contro, XML Schema è estremamente più verboso, tre o quattro volte più lungo del corrispondente DTD, e molto spesso meno chiaro da leggere.
  - In termini computazionali, la validazione è più dispendiosa rispetto ai DTD (usare solo quando serve davvero)

XML Schema Definition (XSD): esempio

```
<?xml version="1.0"?>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       targetNamespace="http://www.dominio.com"
       xmlns="http://www.dominio.com"
       elementFormDefault="qualified">

       <xsd:element name="corso" >
        <xsd:complexType>
          <xsd:sequence>
             <xsd:element name="nome" type="xsd:string"/>
             <xsd:element name="docente" type="xsd:string"/>
             …
          </xsd:sequence>
       </xsd:complexType>
       </xsd:element>
    </xs:schema>
```

<!ELEMENT corso (nome, docente) >
<!ELEMENT nome (#PCDATA)>
<!ELEMENT docente (#PCDATA)>

XSD e namespace

- XSD, essendo un linguaggio XML fa anch'esso riferimento ad un namespace (http://www.w3.org/2001/XMLSchema) e quindi ad uno schema

  <xsd:schema xmlns:xsd= "http://www.w3.org/2001/XMLSchema">
  </xsd:schema>

- Uno schema è una collezione di tipi ed elementi appartenenti ad uno specifico namespace, detto *target namespace*
- La validità di un documento deve essere verificata in relazione ad uno o più schemi
- L'autore dello schema può specificare se gli elementi e gli attributi devono essere qualificati tramite prefissi o tramite il namespace di default

Struttura di un XML Schema

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.dominio.com"
   xmlns="http://www.dominio.com"
   elementFormDefault="qualified">
... ...
</xs:schema>
```

xmlns:xs assegna un nome ad un namespace

L'attributo targetNamespace specifica il namespace associato alle componenti dello schema

Gli attributi elementFormDefault ed attributeFormDefault permettono di controllare se l'uso del prefisso è necessario per gli elementi (attributi) non globali

Riferimento ad uno schema XML

```
<?xml version="1.0"?>
   <c:corso xmlns:c="http://www.dominio.com"
   xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
   xs:schemaLocation="http://www.dominio.com corso.xsd">

      <c:nome> Tecnologie Web</c: nome>
      <c:docente>
            Ombretta Gaggi
      </c:docente>
   </c:corso>
```

xmlns:xs introduce il namespace di XML Schema

Adesso è possibile utilizzare l'attributo schemaLocation

Questi tag sono definiti nel namespace http://www.dominio.com

Local e global scope

- Gli attributi elementFormDefault e attributeFormDefault controllano se gli elementi e gli attributi locali devono essere per default qualificati o non qualificati.
- Per default sono NON qualificati, il che è ragionevole per gli attributi, ma è un po' una sorpresa per gli elementi.

Esempio 1

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns="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>
```

Non funziona!

Esempio 2

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns="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>
```

Neanche questo funziona!

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

Esempio 3

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns="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>
```

Funziona!

```
<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>
```

Esempio 4

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns="http://www.betterbooks.org"
        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>
```

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

Attenzione…

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

Uno schema può contenere:

- <import> ed <include> per inserire, in varia forma, altri frammenti di schema da altri documenti
- <annotation> per esprimere commenti per esseri umani (tag documentation) o per applicazioni (tag appInfo) diverse dal parser di XML Schema.
- <notation> per definire notazioni non XML all'interno di un documento XML

Definizione degli elementi

- Un elemento di definisce così:

```
<xsd:element name="corso" type="TipoCorso"/>

<xsd:element name="nome" type="xsd:string"/>

<xsd:element name="docente" type="xsd:string"/>
```


I tipi in XML Schema

- I tipi sono un modo per esprimere vincoli sul contenuto di elementi ed attributi.
- XML Schema definisce tipi semplici e tipi complessi
- Un tipo semplice non può contenere markup e non può avere attributi. In pratica è una sequenza di caratteri.
  - E' una specifica (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
  - Può avere contenuto semplice o complesso

Tipi semplici

- XML predefinisce un grande numero di tipi semplici: string, decimal, float, boolean, uriReference, date, time, ecc, che possono essere usati sia per attributi che per elementi
- Il nome di un tipo semplice predefinito appartiene allo stesso namespace di XML Schema.
  - <xsd:element name="CCS" type="xsd:string"/>
  - <xsd:attribute name="presidente" type="xsd:string"/>
- È possibile definire nuovi tipi per derivazione dai tipi semplici per restrizione, unione o lista.

Una lista parziale di tipi semplici

- string
- boolean
- integer: una stringa di numeri con segno (ex: +65321)
- decimal: una stringa di numeri (con segno e punto): '-34.15 '
- float: un reale in notazione scientifica: '12.78E-12 '.
- duration: es: 'P1Y2M3DT10H30M ' è la durata di 1 anno, 2 mesi, 3 giorni, 10 ore, e 30 minuti.
- date: nel formato CCYY-MM-DD ('2001-04-25 ‘).
- time: nel formato hh:mm:ss con fuso orario opzionale(ex: '13:20:00+01:00' significa 1:20 PM in MET (+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 spazi bianchi con le stesse proprietà e vincoli di ID e IDREF nei DTD.

Derivazione per restrizione

- Si parte da un tipo già definito e se ne restringe il set di valori leciti:

```
<xsd:element name=“periodo" type=“Ttrimestre"/>
 <xsd:simpleType name="Ttrimestre">
       <xsd:restriction base="xsd:string">
        <xsd:enumeration value="primo trimestre"/>
        <xsd:enumeration value="secondo trimestre"/>
        <xsd:enumeration value="terzo trimestre"/>
       </xsd:restriction>
   </xsd:simpleType>

<xsd:element name="data" >
  <xsd:simpleType >
    <xsd:restriction base="xsd:date">
      <xsd:minInclusive value="2002-01-01"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```


Facets

- Per ogni tipo posso precisare dei facets, cioè delle caratteristiche indipendenti tra di loro che impongono dei vincoli ai valori accettati per quel tipo:
  - length, minLength, maxLength: numero richiesto, minimo e massimo di caratteri
  - minExclusive, minInclusive, maxExclusive, maxInclusive: valore massimo e minimo, inclusivo ed esclusivo
  - totalDigits, fractionDigits: numero massimo di cifre totali e numero di massimo cifre frazionarie
  - pattern: espressione regolare che il valore deve soddisfare
  - enumeration: lista di valori tra cui scegliere il valore voluto(simile alla lista di valori leciti degli attributi nei DTD).
  - period, duration, encoding, ecc.

Restrizione tramite pattern

- Utilizza una sintassi apposita per descrivere i valori ammissibili tramite espressioni regolari
  - a? , a
  - a+: a, aa, aaaaa, …
  - a* : , a, aa, aaaa, …
  - [abcd]/(a|b|c|d) a, b, c, d
  - [a-z] a, b, c, …, z
  - a{2,4} aa, aaa, aaaa
  - ^[0-9]+ sequenza di non cifre

```
<xsd:simpleType name=“telefono”>
  <xsd:restriction base=“xsd:string”>
    <xsd:pattern value=“(0039-)?0[0-9]{2,3}-[0-9]+”/>
  </xsd:restriction>
</xsd:simpleType>
```


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 spazi 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 i tipi semplici), denominati o anonimi (interni alla definizione di un elemento)

```
<xsd:element name=“periodo">
 <xsd:simpleType >
       <xsd:restriction base="xsd:string">
        <xsd:enumeration value="primo trimestre"/>
        <xsd:enumeration value="secondo trimestre"/>
        <xsd:enumeration value="terzo trimestre”/>
       </xsd:restriction>
   </xsd:simpleType>
</xsd:element>
```


Elementi di XML

- Ci sono 4 tipi di elementi in XML:
  - elementi vuoti
  - elementi che contengono solo caratteri
  - elementi che contengono solo altri elementi
  - elementi che contengono sia testo che altri elementi
- Esempi:
  - <targa val=“AA 999 BB” />
  - <ricetta> impastare le uova con la farina… </ricetta>
  - <auto><marca val=“Fiat”/><targa val=“AA 999 BB” /></auto>
  - <torta> <ingredienti> uova, farina,…</ingredienti> impastare le uova con la farina… </torta>

Tipi complessi

- I tipi complessi servono per definire elementi:
  - strutturati (che contengono altri elementi)
  - che contengono sia testo che altri elementi (content model misti)
  - con attributi
- Non esistono tipi complessi predefiniti (o quasi).
- La derivazione può avvenire per restrizione o estensione.

Definizione di un tipo complesso

```
<xsd:element name="corso" type="TipoCorso"/>

  <xsd:complexType name="TipoCorso">
    <xsd:sequence>
      <xsd:element name="nome" type="xsd:string"/>
      <xsd:element name="docente" type="xsd:string"/>
      <xsd:element name="esercitatore" type="xsd:string"/>
      <xsd:element name="CCS" type="xsd:string"/>
      <xsd:element name="periodo" type="Tperiodo"/>
      <xsd:element name="crediti" type="xsd:integer"/>
      <xsd:element name="sitoWeb" type="xsd:anyURI"/>
    </xsd:sequence>
  </xsd:complexType>
```


Tipi complessi - sequence

- Come nei DTD si usano , e | per specificare sequenze e scelte tra gli elementi di un tipo complesso, così in XML schema si usano <sequence> , <choice> e <all>.
  - La sequenza (A, B, C) diventa

```
<xsd:complexType>
  <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>
</xsd:complexType>
```


Tipi complessi - choice

- 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>
```


Tipi complessi - all

- 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).

Tipi complessi - annidamento

- Il raggruppamento non ha bisogno di parentesi ma di annidamento: (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>
```


Cardinalità - 1

- Per default ogni elemento compare una sola volta
- Per specificare ripetitività e facoltatività, si usano gli attributi minOccurs e maxOccurs: prendono come valore qualsiasi numero intero (unbounded = infinito) e 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"/>
```


Cardinalità - 2

- 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>
```


All vs sequence

```
<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>

<xsd:choice maxOccurs=“3”>
   <xsd:element name=”A” type="xsd:string"/>
   <xsd:element name=”B” type="xsd:string"/>
   <xsd:element name=”C” type="xsd:string"/>
</xsd:choice>
```


All vs sequence

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


XML Schema vs DTD: tipo complesso strutturato

- Tipo complesso strutturato:

```
<!ELEMENT X (A,B,C) >

<xsd:element name=“X” type=“Tx”/>
<xsd:complexType name=“Tx”>
   <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>
</xsd:complexType>
```


XML Schema vs DTD: tipo complesso con attributi

- Tipo complesso strutturato con attributi:

```
<!ELEMENT Z (A,B,C) >
<!ATTLIST Z y CDATA (#FIXED z)>

<xsd:element name=“Z” type=“Tz”/>
<xsd:complexType name=“Tz”>
    <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>
     <xsd:attribute name=“y” type=“xsd:string” fixed=“z”/>
</xsd:complexType>
```


Definizione di attributi

- Gli attributi vengono definiti tramite il tag attribute:

```
<xsd:attribute name="attributoFisso" type="xsd:string"
fixed="valore"/>
```

- Gli attributi che si possono definire sono:
  - name
  - type
  - use: required, optional (default)
  - fixed
  - default

Esempio

- Come avviene con i DTD i costrutti possono combinarsi per formare espressioni regolari

<!ELEMENT sezione (titolo, (sottotitolo | abstract)?, para+)>

```
<xsd:element name="sezione">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="titolo" type="xsd:string"/>
   <xsd:choice minOccurs="0">
    <xsd:element name="sottotitolo" type="xsd:string"/>
    <xsd:element name="abstract" type="xsd:string"/>
   </xsd:choice>
   <xsd:element name="para" type="xsd:string"
                                    maxOccurs="unbounded" />
  <xsd:sequence>
 </xsd:complexType>
</xsd:element>
```


Elementi qualsiasi o vuoti

- Si costruiscono sulla base dei tipi predefiniti xsd:anyType e xsd:complexType
- Un elemento qualsiasi (ma definito nello schema) ha tipo xsd:anyType
- Un elemento vuoto è definito con tipo complexType per cui non si specifica nessun componente

Esempio

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

<xsd:element name="br">
   <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="empty" type="emptyTypeWithAttributes"/>
```


ANY e anyAttribute

```
<xsd:complexType name="Tpersona">
    <xsd:sequence>
        <xsd:element name="nome" />
        <xsd:element name= "cognome"/>
        <xsd:any minOccurs="0“ maxOccurs=“unbounded”/>
    </xsd:sequence>
</xsd:complexType>
```

- Mi permette di aggiungere successivamente attributi e elementi non definiti nello schema

Esempio

```
<esercitatore>
    <nome>Ombretta</nome>
    <cognome>Gaggi</cognome>
</esercitatore>

<docente>
    <nome>Massimo</nome> <cognome>Marchiori</cognome>
    <c:sitoWeb>
        http://www.math.unipd.it/~massimo/
    </c:sitoWeb>
</docente>
```


XML Schema e namespace

- Oltre all'utilizzo visto all'inizio, è possibile 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>
```


XML Schema e namespace: valori

- Nell'attributo namespace dell'elemento <any> si può specificare o un namespace vero e proprio, o i valori:
  - ##any: qualunque XML ben formato (default)
  - ##local: qualunque XML non sia qualificato (cioè privo di dichiarazione di namespace)
  - ##targetNamespace: il namespace obiettivo
  - ##other: qualunque XML tranne il target namespace
- L'attributo processContents="skip" viene usato per dire al validatore di non processare quegli elementi (altri valori strict e lax)

Content model misti

- Il content model misto aggiunge semplicemente l'attributo mixed con valore "true" e permette di avere elementi immersi in testo semplice.
- Qualunque espressione di elementi viene rispettata, ma il PCDATA può comparire ovunque, prima o dopo questi elementi.

Esempio 1

```
<!ELEMENT testo (#PCDATA | bold | italic)*>
```

```
<xsd:complexType name="TipoTesto" mixed="true">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
     <xsd:element name="bold" type="xsd:string"/>
     <xsd:element name="italic" type="xsd:string"/>
  </xsd:choice>
</xsd:complexType>

<xsd:element name="testo" type="TipoTesto"/>
```


Esempio 2

```
<letter>
   Dear Mr.<name>John Smith</name>.
    Your order <orderid>1032</orderid> will be shipped on
   <shipdate>2001-07-13</shipdate>.
</letter>
```

Corretto?

```
<xsd:complexType name="letteraOrdini" mixed="true">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
     <xsd:element name="name" type="xsd:string"/>
     <xsd:element name="orderid" type="xsd:integer"/>
     <xsd:element name="shipDate" type="xsd:date"/>
  </xsd:choice>
</xsd:complexType>

<xsd:element name="letter" type="letteraOrdini">
```


Esempio più complesso

```
<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 (con contenuto semplice).
- Questa è la definizione di un tipo il cui contenuto è semplice ma che prevede un attributo.

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

<prezzo valuta=“euro”>15.50</prezzo>
```

Attenzione!

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

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

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 dopo quelli precedentemente definiti.

Derivazione per restrizione

```
<xsd:complexType name ="Tlaureato">

        <xsd:sequence>
            <xsd:element name=“titolo“                    …/>
            <xsd:element name="nome" … />
            <xsd:element name="cognome“ … />
        </xsd:sequence>

</xsd:complexType>
```

Altro esempio

- Il TipoTesto è un testo con elementi bold o italic

```
<xsd:complexType name="TipoTesto" mixed="true">
   <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="bold" type="xsd:string"/>
      <xsd:element name="italic" type="xsd:string"/>
   </xsd:choice>
 </xsd:complexType>

<xsd:element name="testo" type="TipoTesto"/>
```


Altro esempio

- Il TipoTestoConEffetti è un testo con almeno un elemento bold o italic

```
<xsd:complexType name="TipoTestoConEffetti" mixed="true">
  <xsd:restriction base="TipoTesto">
   <xsd:choice minOccurs="1" maxOccurs="unbounded">
      <xsd:element name="bold" type="xsd:string"/>
      <xsd:element name="italic" type="xsd:string"/>
   </xsd:choice>
  </xsd:restriction>
</xsd:complexType>

<xsd:element name="testo" type="TipoTesto" />
```


Derivazioni possibili

- Un tipo derivato per restrizione può essere usato ogni volta che ci si aspetterebbe il tipo base
- Le restrizioni possibili sono:
  - Impostare un default per gli elementi
  - Assegnare un valore fisso o specificare il tipo
  - Restringere i minOccurs-maxOccurs

Derivazione per estensione

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

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


Altro esempio

```
<xsd:complexType name="TipoTestoInternazionale"
                                              mixed="true">
    <xsd:complexContent>
        <xsd:extension base="TipoTesto">
          <xsd:sequence>
            <xsd:element name="estratto" type="xsd:string"/>
          <xsd:sequence>
          <xsd:attribute name="language" type="xsd:string"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:element name="testo" type="TipoTesto" />
```

Definire elementi ed attributi

- Si usano gli elementi **<element>** e **<attribute>**.
- Se figli del tag schema sono elementi ed attributi globali (possono essere root di documenti).
- Altrimenti sono usabili solo all'interno di elementi che li prevedono.
- 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, nillable, ecc.: specificano valori fissi, di default e determinano la possibilità di elementi nulli.

Definizioni locali o globali

- 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.

Esempio

- 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' possibile raccogliere gli elementi 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>
```


Gruppi di attributi

- E' possibile raccogliere gli attributi in gruppi:

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

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


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.

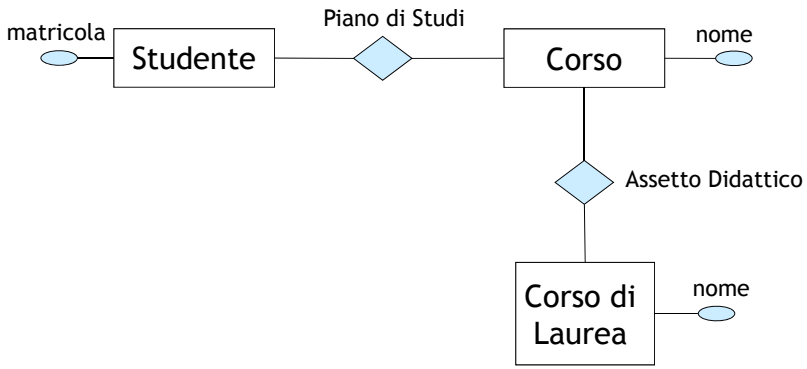
```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.w1.org"
        xmlns="http://www.w1.org"
        xmlns:cd="http://www.w2.org">
  <import namespace="http://www.w2.org"
          schemaLocation="http://www.w2.org/s1.xsd"/>
```


Unicità e chiavi

- In XML Schema, è possibile richiedere che certi valori siano unici, o che certi valori siano chiavi di riferimenti
- Vincolo di Unicità: tag unique
- Chiave primaria: tag key
- Chiave esterna: tag keyref

- Attenzione: la verifica dei vincoli di integrità è estremamente onerosa e deve perciò essere usata solo quando serve

Esempio



Schema Relazionale

- Studente(matricola, nome, cognome)
- Corso(nome, docente, esercitatore, CCS, periodo, …)
- CdL(nome, ...)
- PianoDiStudi(matrStudente, corso)
- AssettoDidattico(CdL, corso)

Schema XML

```
<xsd:unique name="UniStudente">
      <xsd:selector xpath="/CdL/studente" />
      <xsd:field xpath="@matricola" />
</xsd:unique>

<xsd:key name="PKCorso">
     <xsd:selector xpath="/CdL/corso" />
      <xsd:field xpath="nome"/>
      <xsd:field xpath="../@nome"/>
</xsd:key>

<xsd:keyref name="FKCorso" refer="PKCorso">
      <xsd:selector xpath="/CdL/studente/PianoDiStudi" />
      <xsd:field xpath="codCorso"/>
      <xsd:field xpath="../../@nome"/>
</xsd:keyref>
```


Validatori

- On-line
  - W3C http://www.w3.org/2001/03/webdata/xsv
- Da linea di comando:
  - XSV di Henry Thompson ftp://ftp.cogsci.ed.ac.uk/pub/XSV/XSV31.EXE
- Altre informazioni sui tool XML Schema
  - http://www.w3.org/XML/Schema#Tools

Riferimenti bibliografici

- W3Schools, XML Schema Tutorial, http://www.w3schools.com/schema/default.asp
- D. C. Fallside, XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001, http://www.w3.org/TR/xmlschema-0/
- 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/
- P. V. Biron, A. Malhotra,XML Schema Part 2: Datatypes, W3C Recommendation 02 May 2001, http://www.w3.org/TR/xmlschema-2/