XML Schema

Dario Bottazzi
Tel. 051 2093541,
E-Mail: dario.bottazzi@unibo.it,
SkypeID: dariobottazzi

XML Schema Definition (XSD)

- Alternativa ai DTD basata su XML
- Uno **XML Schema descrive la struttura** di un **documento** definendo:
  - **Elementi**
  - **Attributi**
  - Quali elementi sono elementi **figli**
  - **L'ordine** e il **numero** degli elementi **figli**
  - Se un **elemento** è **vuoto**, oppure **contiene testo** o altri **elementi**
  - **Tipi di dati** per elementi e attributi
  - **Valori** fissi o di **default** per **elementi** e **attributi**

XSD vs DTD

Gli schemi XML (**XSD**):

- **Sono** in formato **XML** quindi **possono essere analizzati** da un **parser XML**
- Supportano ***Data Types* primitivi** e **consentono** di **crearne** di nuovi
- Supportano i ***namespace***
- Supportano **l'*ereditarietà*** dei **tipi** ed il ***polimorfismo***

Supporto per *Data Type*

- È possibile **descrivere** il **contenuto** in maniera puntuale → **integer, float, date, string, …**
- È possibile lavorare in modo sicuro con dati estratti da DB → ***Strong Typing***
- È **semplice** la **definizione** di **restrizioni** sui dati
  - → espressioni regolari, enumerativi, numero caratteri, intervalli numerici, …

Estendibilità

- Creazione di **tipi di dato personalizzati** tramite **derivazione** dai **tipi** di dato **disponibili**
- **Utilizzo di più schemi** per la **validazione** di un **singolo documento**
- **Riutilizzo** di **schemi** in altri schemi

Esempio: Messaggio (1)

```
<?xml version="1.0"?>
<message>
  <to>Bob</to>
  <from>Janet</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend</body>
</message>
```

Cosa deve specificare lo schema?

L'elemento **message** è composto di:

1. Elemento **to** contenente una stringa
2. Elemento **from** contenente una stringa
3. Elemento **heading** contenente una stringa
4. Elemento **body** contenente una stringa

Esempio: Messaggio (2)

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="message" type="messageType"/>

    <xs:complexType name="messageType">
      <xs:sequence>
            <xs:element name="to" type="xs:string"/>
            <xs:element name="from" type="xs:string"/>
            <xs:element name="heading" type="xs:string"/>
            <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>

</xs:schema>
```


Who's who?

- Elemento **schema**:
  - Elemento **radice** degli schemi
  - Contiene la **dichiarazione** del ***namespace* degli schemi**
  - Altre dichiarazioni…
- Elemento **element**: **dichiarazione** di **elemento** di nome **name** e di **tipo type**
- Elemento **complexType**: **definizione di tipo** di nome **name**
- Elemento **sequence**: **specifica** del ***content-model*** di **tipo sequenza**

Individuazione schema (1)

Come può il **parser XML individuare** lo **schema** con cui **validare** il documento?

È previsto un "**suggerimento**" che il **parser** può **seguire** o **meno**:

Dichiarazione namespace di specifica del documento istanza

```
<?xml version="1.0"?>
<message
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://mysite.it/msg.xsd">
 <to>Bob</to>
 <from>Janet</from>
 <heading>…</heading>
 <body>…</body>
</message>
```

Dichiarazione ubicazione schema per documento privo di *namespace*

Individuazione schema (2)

Peccato che **normalmente** (dipende dal parser) il "**suggerimento**" **NON** sia **seguito**.

→Per questioni di efficienza **lo schema non** viene **caricato** e il **documento non** viene **validato**

→Per effettuare una validazione **occorre forzare** il **caricamento** dello **schema**

→Ciò che **importa** è il ***namespace*** a cui il **documento** XML **fa riferimento**…

Data Type (1)

Vincolano il **valore** di **elementi** ed **attributi** ad essere di un "**certo tipo**".

Due possibilità:

- **Tipi semplici → valore**
- **Tipi complessi → struttura**

Data Type (2)

- Tipi ***semplici*** (**simpleType**) → valore
  - Tipi ***primitivi***:
    - **Predefiniti** nella specifica XML Schema
    - string, float, integer, positiveInteger, data, …
  - Tipi ***derivati***:
    - **Definiti** in **termini** di **tipi primitivi** (*base*)
    - **Derivazione** per **restrizione**

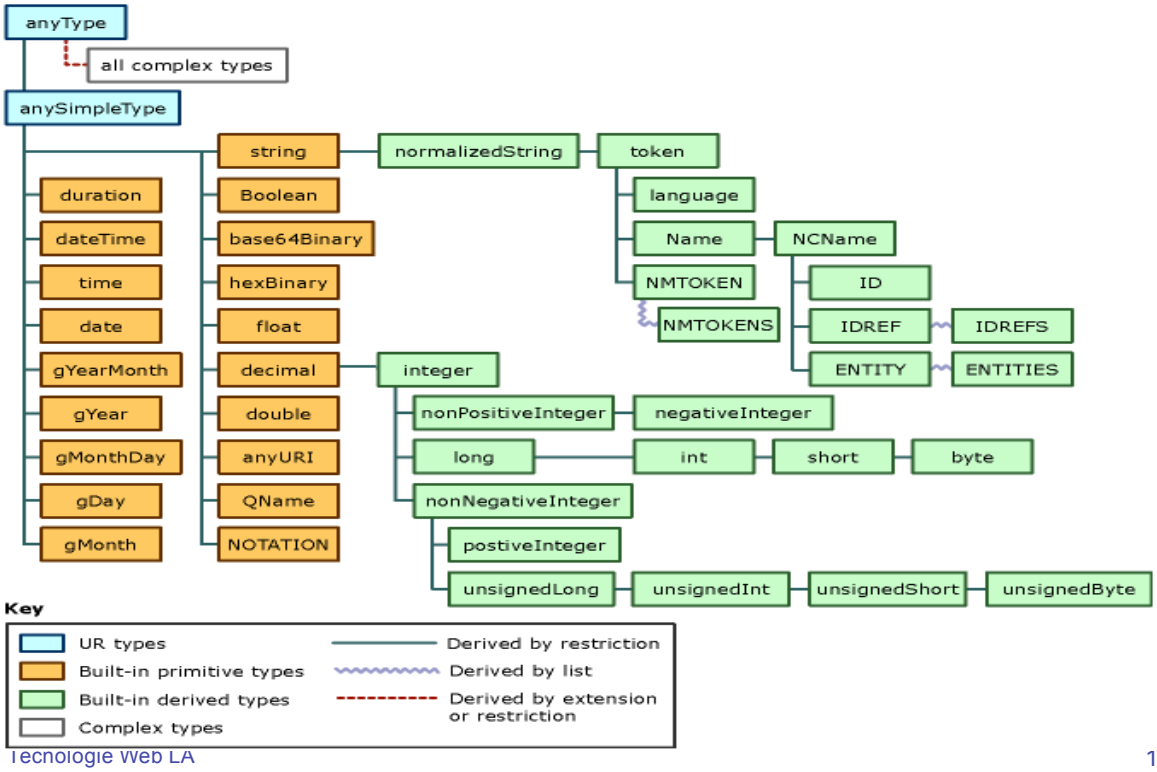
- Tipi ***complessi*** (**complexType**) → struttura
  - **Definizione** di **nuovi tipi** "da zero"
  - **Derivazione** per **estensione** o **restrizione**

- Ovviamente per quanto sin ora detto sull'XML

→ Gli **elementi** possono essere **semplici** o **complessi**

→ Gli **attributi** possono essere **solo semplici**

Data Type – Tassonomia



Definizione vs Dichiarazione (1)

- **Definizione**
  - **Crea** un **nuovo tipo** di dato **semplice** o **complesso**
- **Dichiarazione**
  - Fa **riferimento** ad una **definizione** per **creare** un'**istanza**
  - *La definizione di un tipo può essere inline nella dichiarazione → definizione anonima*

Definizione vs Dichiarazione (2)

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

  <xs:element name="message" type="messageType"/>

  <xs:complexType name="messageType">
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

</xs:schema>
```

Dichiarazione

Dichiarazioni

Definizione

Definizione vs Dichiarazione (3)

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

  <xs:element name="message">

    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>

  </xs:element>

</xs:schema>
```

Dichiarazione

Definizione in-line

Dichiarazione di elementi per tipo (1)

…

<xs:element name="elementName"

type="elementType" />

…

- L'elemento **elementName** è di **tipo elementType** e fa parte del **content-model** di contesto
- **elementType** può essere un **tipo semplice** o un tipo **complesso**

Dichiarazione di elementi per tipo (2)

XSD

```
<xs:element name="Nome"
   type="xs:string" />
<xs:element name="Eta"
   type="xs:positiveInteger" />
<xs:element name="DataNascita"
   type="xs:date" />
```

Tipo semplice predefinito

XML

```
<Nome>Gabriele</Nome>
<Eta>31</Eta>
<DataNascita>1971-10-07</DataNascita>
```

Dichiarazione di elementi per tipo (3)

XSD

```
<xs:element name="Persona" type="PersonaType"/>
<xs:complexType name="PersonaType">
 <xs:sequence>
  <xs:element name="Nome" type="xs:string"/>
  <xs:element name="DataNascita" type="xs:date"/>
 </xs:sequence>
</xs:complexType>
```

Definizione di tipo complesso

XML

```
<Persona>
    <Nome>Gabriele</Nome>
    <DataNascita>1971-10-07</DataNascita>
</Persona>
```


Dichiarazione di elementi per tipo (4)

XSD

```
<xs:element name="Persona">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Nome" type="xs:string"/>
   <xs:element name="DataNascita" type="xs:date"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Definizione inline (o anonima)

A differenza del caso precedente non viene specificato il tipo

XML

```
<Persona>
    <Nome>Gabriele</Nome>
    <DataNascita>1971-10-07</DataNascita>
</Persona>
```

Tipi semplici – Elementi costitutivi

Un **tipo** di dato consiste di:

- **Uno spazio dei valori**
  - Spazio dei valori che un certo tipo di dato può assumere
- **Uno spazio lessicale**
  - Spazio delle rappresentazioni dei valori che un certo tipo di dato può assumere → ***insieme delle stringhe che rappresentano i valori***
- **Un insieme di *facets* (aspetti)**
  - Un ***facet*** è una **proprietà** che **definisce** il **tipo di dato**. **Normalmente** si utilizzano i ***facets*** per **restringere** lo **spazio** dei **valori** del **tipo base** e **creare** un **tipo derivato**

Definizione di tipi semplici (1)

- ***DataType*** di tipo "**valore**"
- Gli elementi dichiarati di questo tipo possono contenere **solo** "**caratteri alfanumerici**" e non altri elementi
- **La definizione di nuovi tipi avviene derivando per restrizione dai tipi predefiniti**
- La restrizione avviene **specificando vincoli (*facets*)** sullo **spazio** dei **valori** o sullo **spazio lessicale**

Definizione di tipi semplici (2)

```
<xs:simpleType name="derivedType">
    <xs:restriction base="baseType">
        <…facets…>
    </xs:restriction>
</xs:simpleType>
```

Tipo derivato

Tipo base

- I ***facets*** – **restrizioni** – **applicabili** **dipendono** dal **tipo base** da cui si deriva.
- È **possibile derivare** anche da **tipi** definiti dall'**utente**
- **Ereditarietà**: un'**istanza** del **tipo** di **dato derivato** verrà **validata correttamente** se **utilizzata al posto** di un'**istanza** del **tipo** di dato **base**

Tipi predefiniti (1)

- **string**: stringa di caratteri esclusi i caratteri di controllo di XML
- **decimal**: numero di **precisione arbitraria** (xxx.yy)
  Tipi derivati:
  - **integer**
  - **positiveInteger**
  - **negativeInteger**
  - …
- **float**: numero **reale** a singola precisione (**32 bit**)

Tipi predefiniti (2)

- **double**: numero reale a doppia precisione (64 bit)
- **boolean**: valore logico true o false
- **dateTime**: rappresenta uno specifico momento temporale.
  Il pattern di base è: **CCYY-MM-DDThh:mm:ss**

  Opzionalmente può comparire un punto decimale per aumentare la precisione dei secondi e, oltre, un'indicazione di *time zone*…

Tipi predefiniti (3)

- **Date**: rappresentazione di una data → v. dateTime
- **Time**: rappresentazione di un'ora → v. dateTime
- Esistono altri tipi che consentono, ad esempio, la rappresentazione di durate temporali, vari tipi di interi con o senza segno, URI, ecc…

Esempio (1.1)

```
<xs:simpleType name="teenAgeType">
  <xs:restriction base="xs:positiveInteger">
    <xs:minInclusive value="13"/>
    <xs:maxInclusive value="19"/>
  </xs:restriction>
</xs:simpleType>
```

Esempio (1.2)

Un elemento o un attributo dichiarato di questo tipo **può assumere valori compresi** fra **13** e **19** estremi **inclusi**.

XSD

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

XML

```
<teenAge>15</teenAge>
```

Esempio (2.1)

```
<xs:simpleType name="minStr">
    <xs:restriction base="xs:string">
        <xs:minLength value="7"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="minMaxStr">
    <xs:restriction base="minStr">
        <xs:maxLength value="14"/>
    </xs:restriction>
</xs:simpleType>
```

minStr è una restrizione di string ed è costituita dalle stringhe di lunghezza non inferiore a 7

minMaxStr è una restrizione di minStr ed è perciò costituita dalle stringhe comprese fra 7 e 14 caratteri.

Equivale a mettere in AND le facets di minStr e di minMaxStr

Un elemento dichiarato di tipo **minMaxStr** può contenere **stringhe** di **lunghezza** variabile **fra 7 e 14**.

Attributo final

- È **possibile impedire** la **derivazione per restrizione da** un **tipo** definito dall'**utente** specificando, nel tipo stesso, l'attributo **final** con **valore restriction**.

```
<xs:simpleType name="minStr" final="restriction">
    <xs:restriction base="xs:string">
        <xs:minLength value="7"/>
    </xs:restriction>
</xs:simpleType>
```

Facet (1)

maxExclusive – minExclusive
maxInclusive – minInclusive

Applicabili a **tutti** i **valori numerici** compresi dateTime, duration, ecc.

È un **errore** se **maxExclusive** compare **insieme** a **maxInclusive** o se **minExclusive** compare **insieme** a **minInclusive**

Vanno in **AND** con **altri *facet*** sia se presenti in uno stesso step di derivazione, sia se presenti in step diversi

Esempio (1.1)

Voto – Da diciotto a trenta…

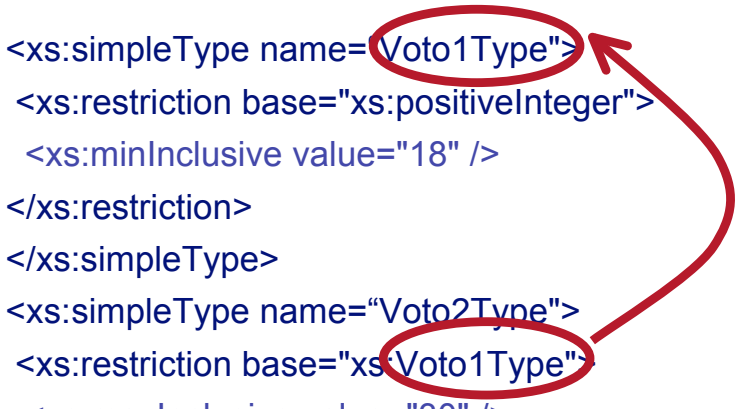
```
<xs:simpleType name="VotoType">
 <xs:restriction base="xs:positiveInteger">
  <xs:minInclusive value="18" />
  <xs:maxInclusive value="30" />
 </xs:restriction>
</xs:simpleType>
```

Sia V istanza di VotoType:
18 <= V && V <= 30

Esempio (1.2)

Voto – Da diciotto a trenta…

```
<xs:simpleType name="Voto1Type">
 <xs:restriction base="xs:positiveInteger">
  <xs:minInclusive value="18" />
</xs:restriction>
</xs:simpleType>
<xs:simpleType name="Voto2Type">
 <xs:restriction base="xs:Voto1Type">
  <xs:maxInclusive value="30" />
 </xs:restriction>
</xs:simpleType>
```



Voto2Type è ottenuto come restrizione di Voto1Type che a sua volta è restrizione di positiveInteger

Equivale all'AND delle Facets

Sia V istanza di Voto2Type:

18 <= V && V <= 30

Facet (2)

length

maxLength – minLength

Applicabili a tutti i valori di tipo **stringa** e derivati ed **anche** a **hexBinary**, **base64Binary**, ecc.

È un **errore** se **length** compare **insieme** a **minLength** o **maxLength**

Vanno in **AND** con **altri *facet*** sia se presenti in uno stesso step di derivazione, sia se presenti in step diversi

Facet (3)

totalDigits – fractionDigits

Applicabili a **decimal** consentono di **limitare** il **numero** di **cifre totali** e **decimali** di un **valore numerico**

Vanno in **AND** con **altri *facet*** sia se presenti in uno stesso step di derivazione, sia se presenti in step diversi

Esempio (2.1)

Euro – tipo derivato che accetta numeri con al più due cifre decimali

```
<xs:simpleType name="EuroType“ final=“restriction”>
 <xs:restriction base="xs:decimal">
  <xs:fractionDigits value="2" />
 </xs:restriction>
</xs:simpleType>
```

Facet (4)

enumeration

Applicabile a **tutti** i **tipi predefiniti limita** tramite **enumerazione** i valori **assumibili**.

Va in **OR** con altri **enumeration** e in **AND** con **altri *facet*** presenti in uno stesso step di derivazione, in AND se presente in step diversi

Esempio (3.1)

Formati audiovideo

```
<xs:simpleType name="AVType">
 <xs:restriction base="xs:string">
        <xs:enumeration value="VHS" />
        <xs:enumeration value="DVD" />
        <xs:enumeration value="DIVX" />
        <xs:enumeration value="BETAMAX" />
        <xs:enumeration value="MINIDV" />
        <xs:enumeration value="VCD" />
 </xs:restriction>
</xs:simpleType>
```

Abbiamo enumerato i valori possibili

Esempio (3.1)

Formati audiovideo su disco

```
<xs:simpleType name="AVDiscType">
 <xs:restriction base="AVType">
      <xs:enumeration value="DVD" />
      <xs:enumeration value="DIVX" />
      <xs:enumeration value="VCD" />
 </xs:restriction>
</xs:simpleType>
```

AVDiscType è dato dai soli valori specificati nella enumeration. I valori erano valori di AVType

Facet (5)

pattern

Applicabile a **tutti** i **tipi predefiniti**; esprime tramite ***espressioni regolari*** i valori assumibili.

Va in **OR** con altri **pattern** e in **AND** con **altri *facet*** presenti in uno stesso step di derivazione, in AND se presente in step diversi

Espressioni Regolari (1)

- Un'**espressione regolare** è una **sequenza** di **caratteri** che **denota** un **insieme di stringhe**

- **Un'espressione regolare** utilizzata per vincolare uno spazio lessicale **impone** che **solo** le **stringhe** appartenenti **all'insieme identificato rappresentino valori validi** per quello spazio

Espressioni Regolari (2)

- Un'espressione regolare può essere suddivisa in:
  - **caratteri ordinari**: trovano una corrispondenza diretta nelle stringe dell'insieme denotato
  - ***metacaratteri****:* caratteri speciali che assumono caratteristiche di controllo sui caratteri ordinari

- Lista dei ***metacaratteri***:

  [ ] \ { } | ( ) ^ ? + * .

Espressioni Regolari (3)

Più rigorosamente si definiscono:

- **Atomo**: un **carattere** ordinario, un **gruppo** di **caratteri** o un'**espressione regolare fra parentesi tonde**

  → Sono atomi: a, [abc], (abc), (a*b+c?) …

- **Parte**: un **atomo eventualmente seguito** da un **quantificatore**

  → Sono parti: a, a*, a{5,6}, (a*b+), (a*b+)?

Espressioni Regolari (4)

- **Quantificatore** (***Quantifier***): vincola il carattere che lo precede a comparire tante volte quanto indicato

- …l'atomo deve comparire:

  {n}: esattamente n volte

  {m,n}: almeno m e al più n volte

  {0,n}: al più n volte

  {m,}: almeno m volte

Espressioni Regolari (4)

- I quantificatori *, +, ? assumono lo **stesso significato che hanno nei DTD** e sono shortcut di:

  * → {0,}
  + → {1,}
  ? → {0,1}

L'espressione banale

- Un'espressione contenente solo caratteri ordinari (solo atomi) denota un insieme contenente un solo elemento.

  Esempio:

  banana → banana

Esempio (1)

a*b+c? → b, ab, abc, bb, abb, abbc, aabbc, aaabc, aaabbbbbc…

Ovvero:

→Zero o più occorrenze di a (quantificatore *) seguite da una o più occorrenze di b (quantificatore +) seguite da zero o una occorrenza di c (quantificatore ?).

→*È una regola per descrivere le stringhe che ci interessano*

Esempio (2)

Ar(har){1,2}ha → Arharha, Arharharha

Ar*gh → Argh, Arrgh, Arrrrgh, …

Gruppi di caratteri (1)

[caratteri] → denota un **insieme** da cui **possono essere scelti tanti caratteri** quanti **indicati** dal **quantificatore**

Esempio:

[ciao]{4} → ciao, oaic, iaoc, … e **tutti gli anagrammi di "ciao"**

Gruppi di caratteri (2)

[A-Z] → denota un insieme composto da un **intervallo** di **caratteri**

Esempio:

[A-Z]{1,10} → tutte le "parole" da 1 a 10 lettere componibili con i caratteri maiuscoli dell'alfabeto

È possibile concatenare intervalli:

→ [A-Za-z] = tutte le lettere dell'alfabeto

Gruppi di caratteri (3)

È possibile **negare** un **gruppo di caratteri** ovvero **richiedere** che il **carattere** (con la sua occorrenza) **non sia fra quelli indicati**.

[^caratteri o intervallo]

Negazione

[^A-Z]{5} → **Tutte le "parole" di 5 caratteri** che **non contengono le lettere maiuscole dell'alfabeto** → possono contenere simboli, cifre, ecc.

Gruppi di caratteri (4)

Sono definiti ***shortcut*** che rappresentano **gruppi** di **caratteri predefiniti**:

- **\d**: corrisponde a una **qualsiasi cifra decimale**; è equivalente a **[0-9]**
- **\D**: corrisponde a un **qualsiasi carattere che non sia una cifra**; è equivalente a **[^0-9]**
- **\w**: corrisponde a un **qualsiasi carattere alfanumerico**; equivale a **[a-zA-Z0-9_]**
- **\W**: corrisponde a un **qualsiasi carattere non alfanumerico**; equivale a **[^a-zA-Z0-9_]**

Altri *metacaratteri*

. → corrisponde a un carattere qualsiasi

| → separa espressioni regolari in OR (*branch)*

\ → consente di inserire (tramite escape) un metacarattere come carattere ordinario

Esempio (3.1)

Codice fiscale – pattern di validazione

```
<xs:simpleType name="CodFiscType">
 <xs:restriction base="xs:string">
  <xs:pattern
   value="[A-Z]{6}\d{2}[A-Z]\d{2}[A-Z]\d{3}[A-Z]"/>
 </xs:restriction>
</xs:simpleType>
```

Esempio 3.2

Lire – Pattern che valida numeri che terminano per '0' o '5'

```
<xs:simpleType name="LireType">
 <xs:restriction base="xs:positiveInteger">
  <xs:pattern value="\d*[50]"/>
 </xs:restriction>
</xs:simpleType>
```

Tutte le cifre che vogliamo basta che l'ultima cifra sia 0 o 5

Esempio 3.3

Euro – tipo derivato che accetta numeri con esattamente due cifre decimali

```
<xs:simpleType name="StrictEuroType">
 <xs:restriction base="EuroType">
  <xs:pattern value="/d*\./d{2}" />
 </xs:restriction>
</xs:simpleType>
```

Tutte le cifre che vogliamo basta che ci sia il carattere "." seguito da 2 cifre. Per inserire "." ho dovuto usare un carattere di escape

Questa derivazione non s'ha da fare! L'attributo final in EuroType preclude la possibilità di derivare un nuovo tipo!

Facet (6)

whitespace

Indica al processore come trattare i caratteri ***spazio*** (**#x20**), ***tab*** (**#x9**), ***line feed*** (**#xA**), ***carriage return*** (**#xD**) nel tipo di dato derivato.

Può assumere i valori:

- **preserve**: **nessuna operazione**
- **replace**: i caratteri ***tab, line feed, carriage*** **return** vengono **sostituiti** da ***spazi***
- **collapse**: viene effettuata l'elaborazione **Replace**, in **più** le **sequenze** di caratteri ***spazio*** vengono **collassate** in un **unico** carattere **spazio** e i **caratteri spazio** all'**inizio** ed alla **fine** del **valore** vengono **eliminati**

Esempio 4

```
<xs:simpleType name="myStr">
 <xs:restriction base="xs:string">
  <xs:whiteSpace value="collapse" />
 </xs:restriction>
</xs:simpleType>
<xs:element name="S" type="myStr" />
```

<S>Ciao</S> → <S>Ciao</S>

<S> Ciao </S> → <S>Ciao</S>

<S> C i a o </S> → <S>C i a o</S>

Facet (7)

- **Non** tutte le **combinazioni** di ***facet*** **danno** un **XSD valido**!
- In linea di principio **sono legali** le **combinazioni** in cui:

1. **Lo spazio dei valori del tipo di dato derivato è più ristretto rispetto a quello del tipo di base**
2. **Lo spazio dei valori del tipo di dato derivato NON è vuoto**

Non tutte le combinazioni "illegali" vengono rifiutate dal processore!!!

Valori di *default* (1)

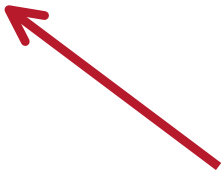
```
<xs:element name="elementName"
            type="elementType"
            default="defaultValue" />
```

- È possibile specificare un **valore** di ***default*** per un **elemento**
- Il valore di *default* viene **utilizzato** **se l'elemento è presente ed è *vuoto***
- **Il valore assegnato deve essere compatibile col tipo di dato**

Valori di *default* (2)

- La definizione di ***vuoto*** **varia** in base al **tipo di dato** nella dichiarazione dell'elemento
- Tutti i **tipi** che **ammettono** come **valore valido** la **stringa vuota non** sono **considerati vuoti**, pertanto il **valore** di ***default*** non è **mai utilizzato**

```
<xs:element name="name"
    type="xs:integer" default="0" />
```



In caso di elemento vuoto viene utilizzato il valore di *default.* Il tipo *integer* non ammette il valore "vuoto"

Valori *fixed*

I **valori fissi** sono inseriti dal processore seguendo le **stesse regole dei valori di *default***. **In aggiunta** se **l'elemento** ha un **valore**, tale valore **deve corrispondere** al **valore fisso dichiarato**.

- **Elemento vuoto** (*tenendo conto del significato di vuoto*) → **valore inserito** dal **processore**
- **Elemento con valore** → il **valore inserito** dal **processore** che **deve corrispondere** al **valore fisso**

ATTENZIONE: **default** e **fixed** sono **mutuamente esclusivi**!!

Valori *nil* (1)

- È **possibile specificare** valori ***nil*** con **significato identico** ai valori **NULL** del mondo dei **database**.
- Nella dichiarazione di elemento **occorre specificare** la **possibilità** di **assumere** il valore ***nil*** **valorizzando** a **true** il **valore** dell'**attributo nillable**
- **Nel** documento **istanza** si **specifica** il valore **nil valorizzando** a **true** l'attributo **xsi:nil** dove **xsi** è il **prefisso** di **namespace** associato all'**URL**: **http://www.w3.org/2001/XMLSchema-instance**

Valori *nil* (2)

Schema

<xs:element name="size" type="xs:integer" nillable="true" />

Istanza

<size xsi:nil="true" /> → Ok

<size xsi:nil="true">10</size> → Errore!

Valori *nil* (3)

- Se **nillable="true" non** è **possibile specificare** un valore **fixed**
- Se **nillable="true"** ed è **specificato** un **valore** di **default**:
  - Se **xsi:nil="true"** il valore di **default non entra in gioco**
  - Se **xsi:nil="false" o non compare**, il **valore** di **default entra in gioco**

Si noti che pur avendo a che fare con simpleType che non supportano attributi, è possibile specificare l'attributo xsi:nil…

Tipi complessi (1)

Gli **elementi** dichiarati di **tipo complesso possono avere attributi** e, in alternativa, **elementi figli** o **contenuto di tipo semplice**.

Gli **attributi** non possono **mai** essere di **tipo complesso**, ma solo di tipo semplice.

Tipi complessi (2)

Tipi di contenuto:

- **Contenuto semplice**:
  solo caratteri e non elementi figli
- **Solo elementi** (specifica di content model, oppure derivazione):
  solo elementi figli e non caratteri
- **Contenuto *mixed:***
  sia caratteri, sia elementi figli
- **Nessun contenuto**

Definizione di tipi complessi

Definizione con nome

```
<xs:complexType name="typeName">
    …tipo di contenuto…
  …attributi…
</xs:complexType>
```

Definizione anonima *inline* con la dichiarazione dell'elemento

```
<xs:element name="myElement">
  <xs:complexType>
    …tipo di contenuto…
    …attributi…
  </xs:complexType>
</xs:element>
```

Solo elementi – Content model

- **sequence**: gli **elementi dichiarati** nella sezione **sequence devono comparire**, nel documento istanza, **nell'ordine** e con le **cardinalità specificate**
- **choice**: degli **elementi dichiarati** nella sezione **choice** ne deve comparire **uno solo** e con la **cardinalità specificata**
- **all**: **tutti** gli **elementi dichiarati** nella sezione **all possono comparire al più una volta** con **ordine qualsiasi**

Sequence (1)

Gli **elementi figli devono comparire** nell'**esatta sequenza** e con le **cardinalità specificate** dagli attributi **minOccur** e **maxOccur**

```
<xs:complexType name="typeName">
 <xs:sequence>
  <xs:element name="e1" minOccur="0"
                    maxOccur="unbounded"
                    type="xs:string" />
  <xs:element name="e2" maxOccur="2"
                    type="xs:string" />
 </xs:sequence>
</xs:complexType>
```

Può non comparire

Al massimo "infinite" volte

Al massimo due volte

Sequence (2)

In un elemento dichiarato di tipo typeName può comparire l'elemento e1 da zero a infinite volte e deve comparire e2 almeno una volta e al massimo due

Dichiarazione:

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

Possibile istanza:

```
<root>
    <e1>Ciao</e1>
    <e1>Riciao</e1>
    <e2>Fine</e2>
</root>
```


minOccur – maxOccur

- **minOccur**: indica il **numero minimo** di **volte** che l'**elemento può comparire**
  Il valore di ***default*** è **1**

- **maxOccur**: indica il **numero massimo** di **volte** che l'**elemento può comparire**
  Il valore di ***default*** **è 1**

Per specificare una **massima cardinalità pari ad infinito** occorre utilizzare la *keyword* **unbounded**

Choice (1)

Gli **elementi figli** devono **comparire** in **maniera mutuamente esclusiva** e con le **cardinalità specificate** dagli attributi **minOccur** e **maxOccur**

```
<xs:complexType name="typeName">
 <xs:choice>
  <xs:element name="e1" minOccur="0"
                    maxOccur="unbounded" />
  <xs:element name="e2" maxOccur="2" />
 </xs:choice>
</xs:complexType>
```

Un elemento dichiarato di tipo typeName può contenere o e1 da zero ad infinite volte (può essere vuoto!) oppure e2 almeno una volta e al massimo due volte

Choice (2)

Un elemento dichiarato di tipo typeName può contenere o e1 da zero ad infinite volte (può essere vuoto!) oppure e2 almeno una volta e al massimo due volte

Dichiarazione:

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

Possibile istanza:

```
<root>
    <e2>Ecco qua</e2>
</root>
```

Sequence – Choice (1)

I gruppi **sequence** e **choice possono**, a loro volta, **contenere** gli attributi di **specifica cardinalità minOccur** e **maxOccur**

```
<xs:complexType name="typeName">
 <xs:sequence minOccur="2" maxOccur="3">
  <xs:element name="e1" type="xs:string"/>
  <xs:element name="e2" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

Quanto meno due volte, e al massimo tre, la sequenza e1, e2

Sequence – Choice (2)

I gruppi **sequence** e **choice possono essere innestati**

```
<xs:complexType name="typeName">
 <xs:sequence>
  <xs:choice>
   <xs:element name="a" type="xs:string"/>
   <xs:element name="b" type="xs:string"/>
  <xs:choice>
  <xs:choice>
   <xs:element name="c" type="xs:string"/>
   <xs:element name="d" type="xs:string"/>
  <xs:choice>
 </xs:sequence>
</xs:complexType>
```

Content model Deterministico (1)

La specifica **XML Schema richiede** che i **modelli** di **contenuto** siano **deterministici**

→ Un **processore** XML Schema deve essere in grado di **individuare** il **ramo corretto** di **validazione senza** dover guardare **avanti** nel **documento istanza**

Content model Deterministico (2)

```
<xs:complexType name="AoBoEntrambiType">
 <xs:choice>
  <xs:element name="a" type="xs:string"/>
  <xs:element name="b" type="xs:string"/>
  <xs:sequence>
   <xs:element name="a" type="xs:string"/>
   <xs:element name="b" type="xs:string"/>
  </xs:sequence>
 </xs:choice>
</xs:complexType>
```

Modello non deterministico

Quando il processore incontra l'elemento a non sa se deve validarlo contro la prima dichiarazione o contro la seconda senza guardare se c'è anche un elemento b.

Content model Deterministico (3)

Un possibile content model deterministico:

```
<xs:complexType name="AoBoEntrambiType">
 <xs:choice>
    <xs:sequence>
  <xs:element name="a" type="xs:string" />
  <xs:element name="b" type="xs:string"
                    minOccurs="0"/>
    </xs:sequence>
    <xs:element name="b" type="xs:string" />
 </xs:choice>
</xs:complexType>
```


All (1)

- Consente di indicare che **tutti** gli **elementi conformi** a quelli **dichiarati** (**dentro all**) **possono comparire** in **qualsiasi ordine al più una volta**
- Può contenere **solo dichiarazioni** di **elementi**
- **Non può comparire all'interno di altri gruppi** (es: sequence, choice)
- **Non è possibile specificare cardinalità** con minOccur e maxOccur **a livello di gruppo**
- I **valori validi** di **minOccur** e **maxOccur** negli **elementi contenuti nel gruppo** sono rispettivamente **(0,1)** e **1**

All (2)

```
<xs:complexType name="typeName">
 <xs:all>
  <xs:element name="e1" type="xs:string"/>
  <xs:element name="e2" type="xs:string"/>
  <xs:element name="e3" type="xs:string"/>
 </xs:all>
</xs:complexType>
```

In un elemento dichiarato di tipo typeName gli elementi e1, e2, e3 devono comparire e possono essere in qualsiasi ordine.

Content model empty

```
<xs:complexType name="typeName">
</xs:complexType>
```

È sufficiente dichiarare un tipo **complexType privo di contenuto.**

Attributi

Gli **attributi** possono essere contenuti **solo** da elementi di tipo **complexType** e **devono** essere dichiarati **dopo** il modello di contenuto

```
<xs:complexType name="WAttrType">
 <xs:sequence>
   <xs:element name="a" type="xs:string"/>
   <xs:element name="b" type="xs:string"/>
 </xs:sequence>
 <xs:attribute name="at"
               type="xs:string"/>
</xs:choice>
```

Attributi

Dichiarazione di attributi (1)

```
<xs:attribute name="attributeName"
   type="attributeSimpleType"
   use="optional|prohibited|required"/>
```

- **name**: nome dell'attributo
- **type**: tipo dell'attributo (solo simpleType)
- **use**:
  - **optional**: l'attributo può non comparire
  - **prohibited**: l'attributo NON deve comparire
  - **required**: l'attributo DEVE comparire

Dichiarazione di attributi (2)

```
<xs:attribute name="attributeName"
              use="optional|prohibited|required">
   <xs:simpleType>
      …
   </xs:simpleType>
</xs:attribute>
```

Alternativa: dichiarazione di attributo con definizione anonima di tipo (inline)

Valori default e fixed (1)

```
<xs:attribute name="attrName1"
              type="aType"
              default="value"/>
…
<xs:attribute name="attrName1"
              type="aType"
              fixed="value"/>
```

Valori default e fixed (2)

La **logica** è diversa rispetto agli elementi, ma **identica** rispetto alle **dichiarazioni DTD**:

- **Default**: **se** l'**attributo non** è **presente**, viene **inserito** il valore di ***default***, **altrimenti** il valore di ***default* non entra in gioco**
- **Fixed**: **se l'attributo non** è **presente**, viene **inserito** il **valore *fixed***, **altrimenti** il **valore nel documento** istanza **deve essere uguale** al valore ***fixed***

fixed e **default** sono **mutuamente esclusivi**!

Attributi su elementi a contenuto semplice (1)

- Gli **attributi** possono essere dichiarati **solo** su **elementi complessi**
- Occorre un metodo **per** poter **inserire attributi** su **elementi** con **contenuto semplice**

TRUCCO

→ **Si deriva**, per **estensione**, un **tipo complesso** da un **tipo semplice** e gli si **impone** la **presenza** degli **attributi** desiderati

Attributi su elementi a contenuto semplice (2)

```
<xs:complexType name="simpleWAttType">
    <xs:simpleContent>
        <xs:extension base="baseType">
            <xs:attribute name="attName"
                          type="attType"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
```

In una extension per un simpleContent è possibile solamente dichiarare attributi

…possibile dichiarazione di attributo con definizione anonima di tipo (inline)

Dichiarazioni globali (1)

- È **possibile effettuare dichiarazioni globali** di **elementi** ed **attributi** e **referenziare** tali dichiarazioni per **effettuare altre dichiarazioni**.
- Le **dichiarazioni globali** compaiono al ***top level*** dello **schema** quindi come figli diretti dell'elemento schema

  <xs:element ref="aGlobalElement"/>

- L'**elemento** o **attributo dichiarato per riferimento** ha le **stesse proprietà** (nome, tipo, ecc.) dell'**elemento** o **attributo riferito** e tali **proprietà NON possono essere ridefinite**

Dichiarazioni globali (2)

In una **dichiarazione globale NON** è possibile:

- **Utilizzare un riferimento per la dichiarazione**: occorre effettuare dichiarazioni che utilizzino un tipo predefinito o definito dall'utente oppure utilizzare una definizione *inline*
- **Esprimere vincoli di cardinalità negli elementi**: minOccurs e maxOccurs NON possono comparire
- **Esprimere vincoli di uso negli attributi**: *use* NON può comparire

Dichiarazioni globali (3)

Eventuali **vincoli** di **cardinalità** per gli **elementi** o di **uso** per gli **attributi** vanno **eventualmente specificati** nelle **dichiarazioni** di **elemento/attributo** che si **riferiscono** (usano l'attributo *ref*) all'**elemento/attributo globale** per la loro dichiarazione.

Dichiarazioni globali – Esempio (1)

```
<xs:schema …>
  <xs:element name="comment"
              type="xs:string"
              default="none"/>
    <xs:attribute name="att"
              type="xs:integer"
              nillable="true"/>
    <xs:element name="myRoot">
      <xs:complexType>
          <xs:sequence>
            <xs:element name="a" type="xs:float"/>
            <xs:element ref="comment"/>
          </xs:sequence>
          <xs:attribute ref="att"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```


Dichiarazioni globali – Esempio (2)

Possibile documento istanza:

```
<myRoot att="5">
   <a/>
   <b/>
   <comment>Sono nato stanco!</comment>
</myRoot>
```

Possibile documento istanza:

```
<comment>No comment</comment>
```

Dichiarazioni globali (4)

Più dichiarazioni globale di elemento → i possibili **documenti istanza** possono avere **radici di tipi diversi**!

Possibilità di validare frammenti di documento…

- …per inserirli in un documento più "grande"
- …perché validare tutto se ne serve solo una parte?
- …

Contenuto mixed

- Stesso significato che ha in DTD → **consente la presenza di caratteri e di elementi**
- Ha senso parlare di contenuto mixed **solo per tipi complessi**
- La specifica di **contenuto mixed non** è **alternativa alla specifica** di un **modello di contenuto** come in DTD
- → il **modello di contenuto DEVE essere rispettato**
- → *mixed* di DTD e *mixed* di XML Schema NON SI EQUIVALGONO!

Contenuto mixed – Esempio (1)

```
<xs:schema …>
 <xs:complexType name="ClienteType">
  <xs:sequence>
     <xs:element name="nome" type="xs:string"/>
     <xs:element name="cognome" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="LetterType" mixed="true">
  <xs:sequence>
     <xs:element name="cliente" type="ClienteType"/>
     <xs:element name="prodotto" type="xs:string"/>
     <xs:element name="taglia" type="xs:positiveInteger"/>
  </xs:sequence>
 </xs:complexType>
 <xs:element name="letter" type="LetterType"/>
</xs:schema>
```


Contenuto mixed – Esempio (2)

```
<letter>
   Sono <cliente>
                <nome>Gabriele</nome>
                <cognome>Zannoni</cognome>
          </cliente>
   e compro un <prodotto>teletrasporto</prodotto> taglia
   <taglia>2</taglia>
</letter>
```

- **La sequenza degli elementi DEVE essere rispettata!!**
- **Cliente non è *mixed* quindi non può contenere caratteri**

Namespace (1)

- Uno **schema** = insieme di **definizioni** e **dichiarazioni** i cui **nomi appartengono** ad un particolare ***namespace*** (*targetNamespace*)
- Ogni documento schema può avere ***un solo targetNamespace***
- L'attributo ***targetNamespace*** va posto nell'**elemento schema radice**
- Il ***namespace*** cui fa **riferimento** il **targetNamespace deve essere dichiarato**

Namespace (2)

- XML Schema è un linguaggio XML → con l'utilizzo di *namespace* è **possibile distinguere** i **nomi** del **vocabolario XML Schema** dai **nomi del vocabolario** che si sta **definendo**
- In un documento XMLSchema che **vuole** "**definire**" un ***namespace* occorre dichiarare almeno due *namespace***:
  - Il namespace degli schemi: **http://www.w3.org/2001/XMLSchema**
  - Il namespace obiettivo

Namespace (3)

```
<schema
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:po="http://example.com/PO1"
  targetNamespace="http://example.com/PO1"
  ... >
...
</schema>
```

Tutte le **definizioni** e le **dichiarazioni** all'**interno** del **documento faranno parte** del ***namespace* obiettivo**

Namespace (4)

Lo schema si riferisce ad un namespace

→ per "**suggerire**" al parser **dove individuare** lo **schema**:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=
        "namespaceURI1 schemaURL1
         namespaceURI2 schemaURL2
         ... ..."
```

Usare l'attributo **schemaLocation** al posto di **noNamespaceSchemaLocation**

Qualificazione (1)

Si può decidere se gli **elementi** e gli **attributi locali debbano** essere **qualificati** da un **prefisso** nel **documento istanza**.

Locali sono le **dichiarazioni** effettuate **tramite tipo** o direttamente **inline** e **NON per riferimento**

Per stabilire un comportamento di default:

elementFormDefault="unqualified | qualified"
attributeFormDefault="unqualified | qualified“

Sono attributi dell'elemento schema; il loro default è **unqualified**

Qualificazione (2)

Per stabilire un comportamento "locale" che sovrascrive il default:

form="unqualified | qualified“

Attributo da porre nella dichiarazione di elemento o di attributo

```
<xs:complexType name="myType">
    <xs:sequence>
        <xs:element name="myEl" type="aSimpleType"
                form="unqualified" />
    </xs:sequence>
</xs:complexType>
```

Unqualified locals – Esempio (1)

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:po="http://example.com/PO1"
  targetNamespace="http://example.com/PO1"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified">

<element name="purchaseOrder" type="po:POType"/>

<element name="comment"  type="string"/>
```

Dichiarato per riferimento vedi slide successiva

Unqualified locals – Esempio (2)

```
<complexType name="POType">
 <sequence>
  <element name="shipTo" type="po:Address"/>
  <element name="billTo" type="po:Address"/>
  <element ref="po:comment" minOccurs="0"/>
 </sequence>
 <attribute name="orderDate" type="date"
            use="required"/>
</complexType>
```

Unqualified locals – Esempio (3)

```
<complexType name="Address">
 <sequence>
  <element name="name"   type="string"/>
  <element name="street" type="string"/>
  <!-- etc. -->
 </sequence>
</complexType>

</schema>
```

Commento SGML → vale anche per XML, HTML, XHTML, …

Unqualified locals – Esempio (4)

```
<?xml version="1.0"?>
<apo:purchaseOrder
  xmlns:apo="http://www.example.com/PO1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.com/PO1 POSch.xsd"
orderDate="2003-10-20">

    <shipTo>
       <name>Alice Smith</name>
       <street>123 Maple Street</street>
       <!-- etc. -->
    </shipTo>
```

Unqualified locals – Esempio (5)

```
    <billTo>
       <name>Robert Smith</name>
       <street>8 Oak Avenue</street>
       <!-- etc. -->
    </billTo>
    <apo:comment>
      Hurry, my lawn is going wild!
    </apo:comment>
    <!-- etc. -->
</apo:purchaseOrder>
```


Unqualified locals – Esempio (6)

- Il **prefisso** associato al **namespace obiettivo** nell'istanza è **diverso rispetto** a **quello associato** nello **schema** (apo != po) → Il fatto che siano diversi è irrilevante!
- L'elemento **comment** è stato **qualificato** poiché **dichiarato per riferimento** → non è una dichiarazione locale, ma **globale** come purchaseOrder!!

→ **Tutte le dichiarazioni globali vanno qualificate**!!!

Qualified locals – Esempio (1)

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:po="http://www.example.com/PO1"
    targetNamespace="http://www.example.com/PO1"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">

… stesse dichiarazioni precedenti …

</schema>
```


Qualified locals – Esempio (2)

```
<?xml version="1.0"?>
<apo:purchaseOrder xmlns:apo="http://www.example.com/PO1"
   orderDate="1999-10-20">
    <apo:shipTo>
       <apo:name>Alice Smith</apo:name>
       <apo:street>123 Maple Street</apo:street>
       <!-- etc. -->
    </apo:shipTo>
    <apo:billTo>
       <apo:name>Robert Smith</apo:name>
       <apo:street>8 Oak Avenue</apo:street>
       <!-- etc. -->
    </apo:billTo>
    <apo:comment>Hurry!!!</apo:comment>
</apo:purchaseOrder>
```

Tutti gli elementi sono qualificati!!

Qualified locals – Esempio (3)

```
<?xml version="1.0"?>
<purchaseOrder xmlns="http://www.example.com/PO1"
    orderDate="1999-10-20">
   <shipTo>
      <name>Alice Smith</name>
      <street>123 Maple Street</street>
      <!-- etc. -->
   </shipTo>
   <billTo>
      <name>Robert Smith</name>
      <street>8 Oak Avenue</street>
      <!-- etc. -->
   </billTo>
   <comment>Hurry!!!</comment>
</purchaseOrder>
```

Tutti gli elementi sono qualificati… per *default*!!

Problema 1 (1)

```
<xs:schema targetNamespace=“myschema.org"
      elementFormDefault=“unqualified"
      xmlns:p=“myschema.org"
      xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="root">
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="a" type="xs:string" />
            </xs:sequence>
      </xs:complexType>
   </xs:element>
   </xs:schema>
```

Produrre un documento istanza valido che utilizzi il prefisso di *default* (il prefisso vuoto)…

Problema 1 (2)

```
<root xmlns="myschema.org">
   <a>Ciao ciao</a>
</root>
```

L'elemento **a** è **automaticamente qualificato**, quindi il **documento non è valido**!!!

→ Non è possibile utilizzare il namespace di default con documenti che richiedono la NON qualificazione degli elementi locali!

Qualificazione di attributi

- **Devono** essere **qualificati** gli **attributi**
  - **Globali**
  - **Con definizione form="qualified"**
  - **Tutti se attributeFormDefault="qualified"**

- **Non esiste per gli attributi** un meccanismo di **qualificazione di *default*** come per gli elementi

→NON è possibile utilizzare il *namespace* di *default* per qualificare automaticamente gli attributi!!