DTD -> lo useremo nel progetto
XML Schema -> nel prossimo laboratorio

# XSDL XML Schema Description Language

DR WIOLETA SZWOCH

DEPARTMENT OF INTELLIGENT INTERACTIVE SYSTEMS

Perchè ne abbiamo bisogno? -> le stesse informazioni (nome, cognome e data) possono essere salvate in più modi. Se ogni persona crea la sua versione è un problema. Bisogna creare delle regole.

## XML Schema

An XML Schema describes the structure of an XML document.

dictionary for XML file

2001 standard XML Schema

W3C XML Schema specification

- Part 0 fundamentals
- Part 1 structures elements, attributes, namespaces
- Part 2 data types

## Purpose of XML Schema

data validation
- elements and attributes structure
- elements order
- values of elements and attributes

system documentation

modifying data

application-specific information

recipe for language

validator control correcness of the document 
the programmer analyzes only the correct documents, he does not have to examine erroneous cases

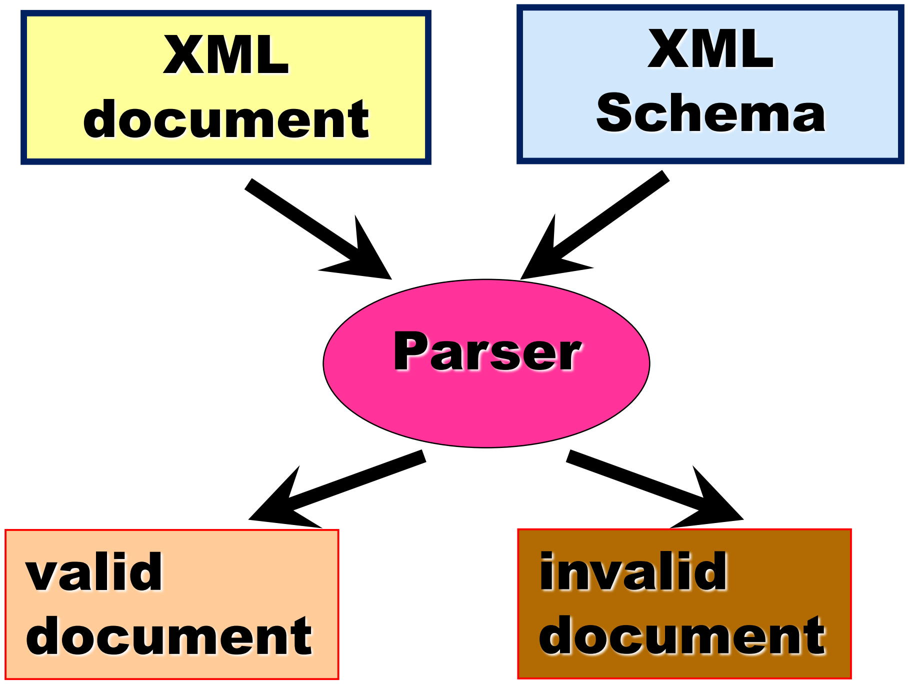
la stessa cosa può essere descritta in più modi con XML schema

Come possiamo controllare se il nostro XML file va bene con l'XML Schema che abbiamo

# Correctness of the document

well-formed document

valid document



# Correctness of the document

## *WELL-FORMED*

There must be exactly one root element

Every start tag has a matching end tag

Attribute values must be quoted

Elements may nest, but must not overlap

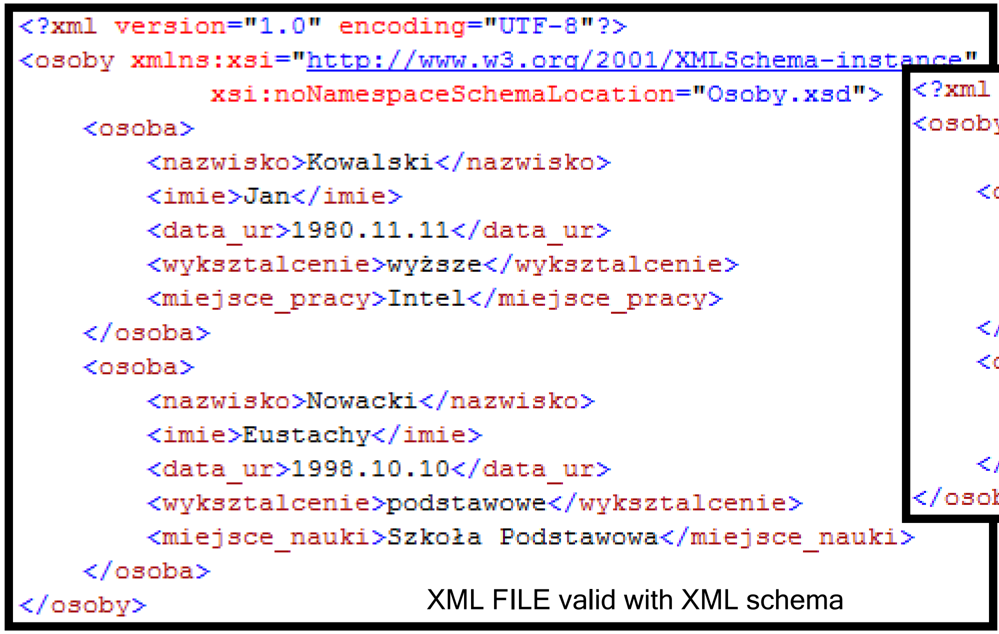
…

## *VALID*

is well- formed

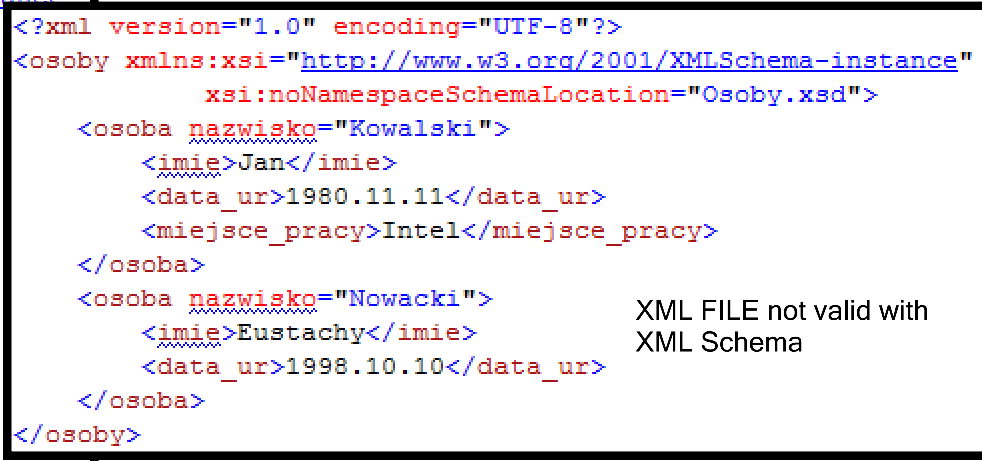
the definition of the document exists (DTD, XML Schema, . . . )

the content of the XML document is consistent with the definition of the document



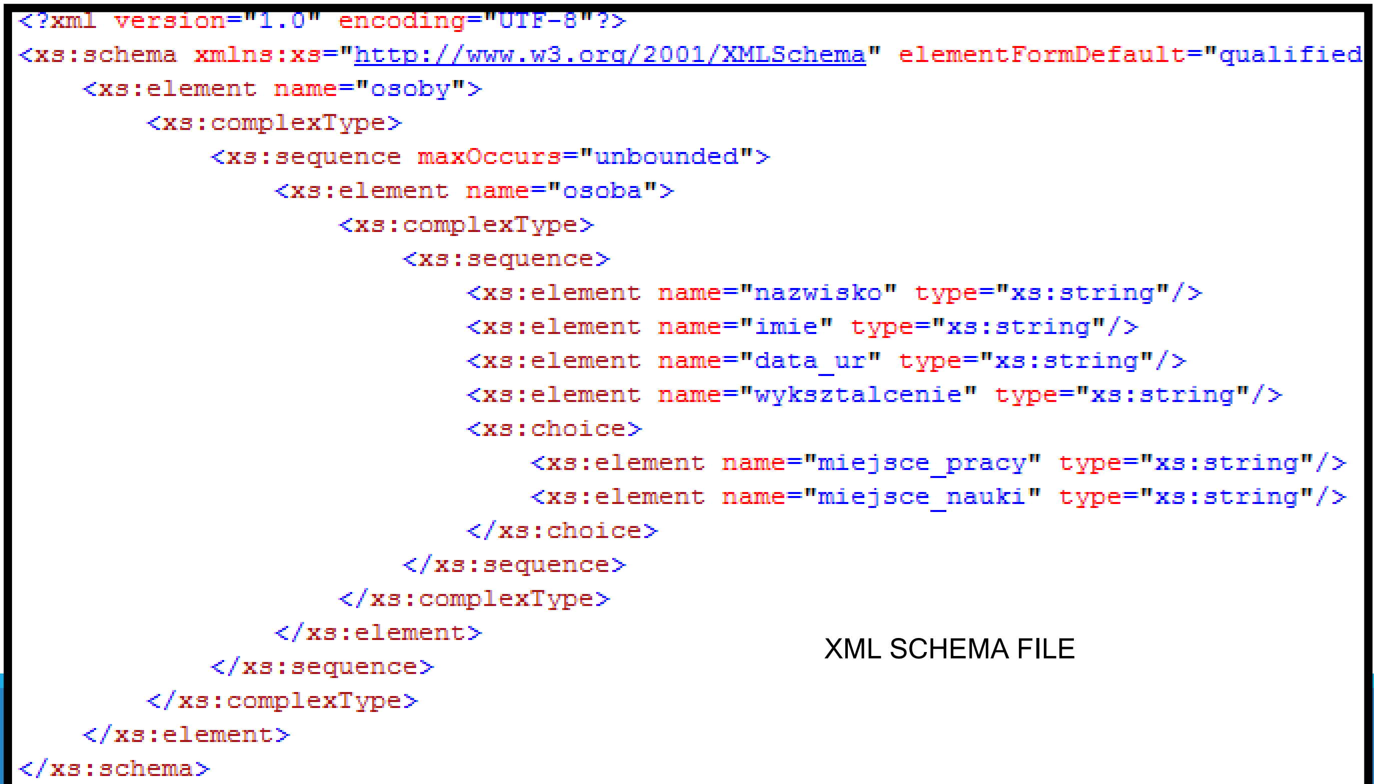
```
<?xml version="1.0" encoding="UTF-8"?>
<osoby xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="Osoby.xsd">
    <osoba>
        <nazwisko>Kowalski</nazwisko>
        <imie>Jan</imie>
        <data_ur>1980.11.11</data_ur>
        <wyksztalcenie>wyższe</wyksztalcenie>
        <miejsce_pracy>Intel</miejsce_pracy>
    </osoba>
    <osoba>
        <nazwisko>Nowacki</nazwisko>
        <imie>Eustachy</imie>
        <data_ur>1998.10.10</data_ur>
        <wyksztalcenie>podstawowe</wyksztalcenie>
        <miejsce_nauki>Szkoła Podstawowa</miejsce_nauki>
    </osoba>
</osoby>
```

XML FILE valid with XML schema



```
<?xml version="1.0" encoding="UTF-8"?>
<osoby xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="Osoby.xsd">
    <osoba nazwisko="Kowalski">
        <imie>Jan</imie>
        <data_ur>1980.11.11</data_ur>
        <miejsce_pracy>Intel</miejsce_pracy>
    </osoba>
    <osoba nazwisko="Nowacki">
        <imie>Eustachy</imie>
        <data_ur>1998.10.10</data_ur>
    </osoba>
</osoby>
```

XML FILE not valid with XML Schema



```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified
    <xs:element name="osoby">
        <xs:complexType>
            <xs:sequence maxOccurs="unbounded">
                <xs:element name="osoba">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="nazwisko" type="xs:string"/>
                            <xs:element name="imie" type="xs:string"/>
                            <xs:element name="data_ur" type="xs:string"/>
                            <xs:element name="wyksztalcenie" type="xs:string"/>
                            <xs:choice>
                                <xs:element name="miejsce_pracy" type="xs:string"/>
                                <xs:element name="miejsce_nauki" type="xs:string"/>
                            </xs:choice>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

XML SCHEMA FILE

## XML Schema

XML Schema is an external file

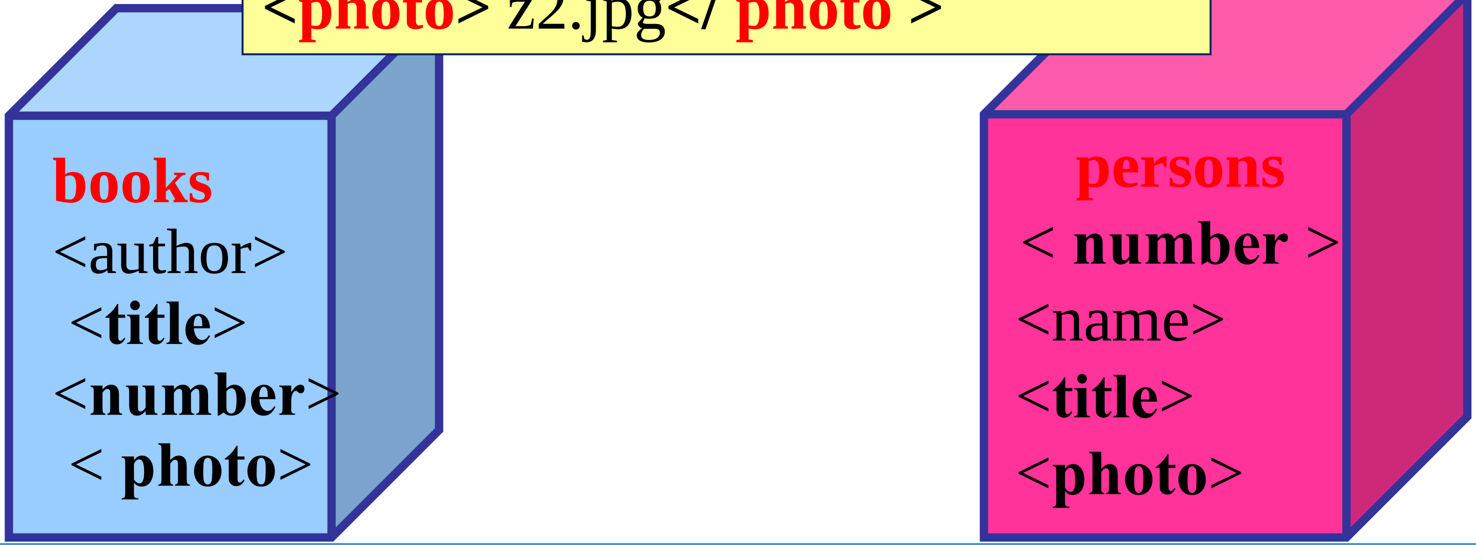
XML Schema defines classes of XML documents

- XML Schema
- XML Data,
- Document Content Description (DCD),
- Schema for Object-oriented XML (SOX)
- Schematron
- TREX
- RELAX, XDR, HOOK, DSD, Assertion Grammars

Problema: se vogliamo unire due file ma che hanno gli stessi attributi non sappiamo come fare a distinguarli. In questo caso abbiamo il problema con titile, number and photo. Putroppo la slide successiva non la fa vedere, ma per capire basta aggiungere prima del elemento in nome de file seguito dai due punti e poi il tag. Esempio: <books:title> ... </books:title>

# XML namespaces

```
<number> 1</ number >
<author> Kowalski</author>
<title> Profesor </ title >
< photo > z1.jpg</ photo >
<name> Nowak</ name>
< title > Profesor </ title >
< number > 1</ number >
<photo> z2.jpg</ photo >
```



# XML namespaces

necessary when we have a name conflict becouse we use different markup languages

Marker language treated as a set of names - namespace

Namespace

- namespace is defined by **xmlns:prefix**
- prefix is used for every tag or attribute from the namespace
- identified by URI
- each tag from the namespace has a prefix

Un namespace si devinisce con **xmlns:"nome_del_namespace"**.
Qui ci sono tutti i casi, penso li capirai con il tempo.

# XML namespaces

namespace in document

- no namespace
- one
- many
- local
- default

```
<?xml version="1.0"?>
<book>
   <title> Digital documents </title>
</book>
```

```
<?xml version="1.0"?>
<bk:book xmlns:bk='http://www.books.org/books'>
   <bk:title> Digital documents </bk:title>
</bk:book>
```

```
<?xml version="1.0"?>
<bk:book xmlns:bk='http://www.books.org/books‘
            xmlns:isbn='urn:ISBN:0-395-36341-6'>
   <bk:title> Digital documents </bk:title>
   <isbn:number>1568491379</isbn:number>   </bk:book>
```

```
<?xml version="1.0"?>
<bk:book xmlns:bk='http://www.books.org/books'>
   <bk:title> Digital documents </bk:title>
   <isbn:number xmlns:isbn='urn:ISBN:0-395-36341-
      1568491379
   </isbn:number>
</bk:book>
```

```
<?xml version="1.0"?>
<book xmlns='http://www.books.org/books'>
   <title> Digital documents </title>
   <isbn:number xmlns:isbn='urn:ISBN:0-395-36341-6'>
      1568491379
   </isbn:number>
</book>
```

Usa gli standard!!!!

## XML standard namespaces

| language | prefiks | URI |
|---|---|---|
| HTML | html: | http://www.w3.org/TR/REC-html40 |
| XML Schema | xsd: | http://www.w3.org/2001/XMLSchema-instance |
| XSLT | xsl: | http://www.w3.org/1999/XSL/Transform |
| XSL | fo: | http://www.w3.org/1999/XSL/Format |
| Xlink | xlink: | http://www.w3.org/1999/xlink |

## XML Schema

XML Schema namespace

- http://www.w3.org/2001/XMLSchema
- there are all Schema components
    - element, attribute, schema, complexType, string, sequence ...

Questi due schemi sono assolutamente identici al 1 ha il namespace di default senza prefisso, il 2 ha il prefisso stardand per XML Schema

## XML Schema

```
<schema xmlns="http://www.w3.org/2001/XMLSchema”>
   <element name=“Person”>
      <complexType>
      ...
      </complexType>
   </element>
   ...
</schema>
```

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema”>
   <xsd:element name=“Person”>
      <xsd:complexType>
      ...
      </xsd:complexType>
   </xsd:element>
   ...
</xsd:schema>
```

XML Schema è pur sempre un file XML quindi anche qui ci sono le solite regole tipo la versione e il root ecc...

# XML Schema

Structure of the XML Schema document

```
<?xml version="1.0" ?>

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

....

</xsd:schema>
```

namespace for XML Schema

the main content of the scheme

# XML Schema

element declaration

- type, numer of elements, hierarchy

File XML, nel file XML Schema dovremo essere in grado di definire elementi e attributi

attribute declaration

new types definition

```
<person surname=‘Nowak’ >
        <name> Anna </name>
        <name> Marta </name>
        <data> 1980.01.01 </data>
</ person>
```

Blu = XML Schema
Yellow = XML

# Element declaration

Element

```
<xsd:element name=”Name_of_the_element”  type=”Data_Type” ... attributes/>
```

```
< Name_of_the_element >
   element content according to Data_Type
</ Name_of_the_element >
```

```
<xsd:element name=”surname”  type=”xsd:string” />
```

```
<surname> Kowalski </ surname>
```

# Elements attributes

- name
- type
- id
- minOccurs
- maxOccurs
- ref

occurrence

```
<xsd:element name=”surname”  type=”xsd:string” />
```

## Element attributes

*minOccurs, maxOccurs*

- default value
  - minOccurs = ”1”
  - maxOccurs = ”1”
- optional
  - minOccurs = ”0”
- Unspecified number of occurrences
  - maxOccurs = ”unbounded” (non ci sono vincoli)

```
<schema xmlns=“http://www.w3.org/2001/XMLSchema>
…
          <element name=“name” type=“string”
                          minOccurs=“0” maxOccurs=“unbounded” />
            <element name=“surname” type=“string”
                          minOccurs=“1” maxOccurs=“1” />
            <element name=“phone” type=“long”
                          minOccurs=“1” maxOccurs=“3” />
…
</schema>
```

## Element declaration

Elements

- Global
- Local
- References

## Global elements

Placed directly in the main element

Messi direttamente nella root del file

Visible in the whole scheme

Sono visibili in tutto lo schema

They have a name

They have a type

They can have optional attributes

```
<FirstGlobalElement>
        content
</ FirstGlobalElement >
```

```
<schema xmlns="http://www.w3.org/2001/XMLSchema>
...
<element name="FirstGlobalElement" type="string" />
...
</schema>
```

# Local elements

They are defined in the context of other elements

They have a name, type, they can have optional attributes

Se un elemento è locale, è invisibile nel resto dello schema

XML SCHEMA

```
<schema xmlns="http://www.w3.org/2001/XMLSchema
   <element name="FirstGlobalElement">
...
      <element name="LocalElement"  type="float" />
...
   </element>
   <element name="SecondGlobalElement" >
...
      <element name="LocalElement"  type="string" />
...
   </element>
</schema>
```

XML FILE (rispetta le condizioni dell'XML SCHEMA

```
< FirstGlobalElement >
      < LocalElement >1.2 </ LocalElement >
</ FirstGlobalElement >
< SecondGlobalElement >
      < LocalElement >plain text </ LocalElement >
</ SecondGlobalElement >
```

We can declare a global element and refer to it in another element

# References to the elements

**They have the ref attribute instead of the name attribute**

They do not have any type attribute

They allow multiple use of element declaration

They can only refer to global elements(?)

```
<schema xmlns=“http://www.w3.org/2001/XMLSchema>
   <element name=“FirstElement” >
...
      <element ref=“SecondElement” />
...
   </element>
...
   <element name=“SecondElement” type=“string” />
</schema>
```

```
<FirstElement>
      <SecondElement>wartość</SecondElement>
</ FirstElement >
```

## Declaration *v.s.* Definition

*Declarations* – enable element and attributes with specific names and types to appear in document instance

- element declaration
- attribute declaration

thanks to declaration elements and attributes with specific name and type can appear in a document istance (in a XML file)

*Definitions* - create a new type

- simple type definitions
- complex type definitions
- attribute group, model group definitions

# Elements - fixed and default value

fixed = come default ma non puoi rimodificarlo

```
<xsd:element name= ” number” fixed= ”1.0” />
```

```
< number >1.0</ number >
```

```
< number />
```

~~< number >2.0</ number >~~

default = valore di default che ha l'attributo se è vuoto

```
<xsd:element name= ” number” type= ”decimal” default= ”1.0” />
```

```
< number >1.0</ number >
```

```
< number />
```

```
< number >2.0</ number >
```

E' un errore avere sia default che fixed

## Data types

built-in = are part of the  standard

user-defined = are types that the user have defined

simple type: when you want to create a new type that is a refinement of a built-it type

complex type = can contain sub element. Use when you want to define child elements and/or attributes of an element

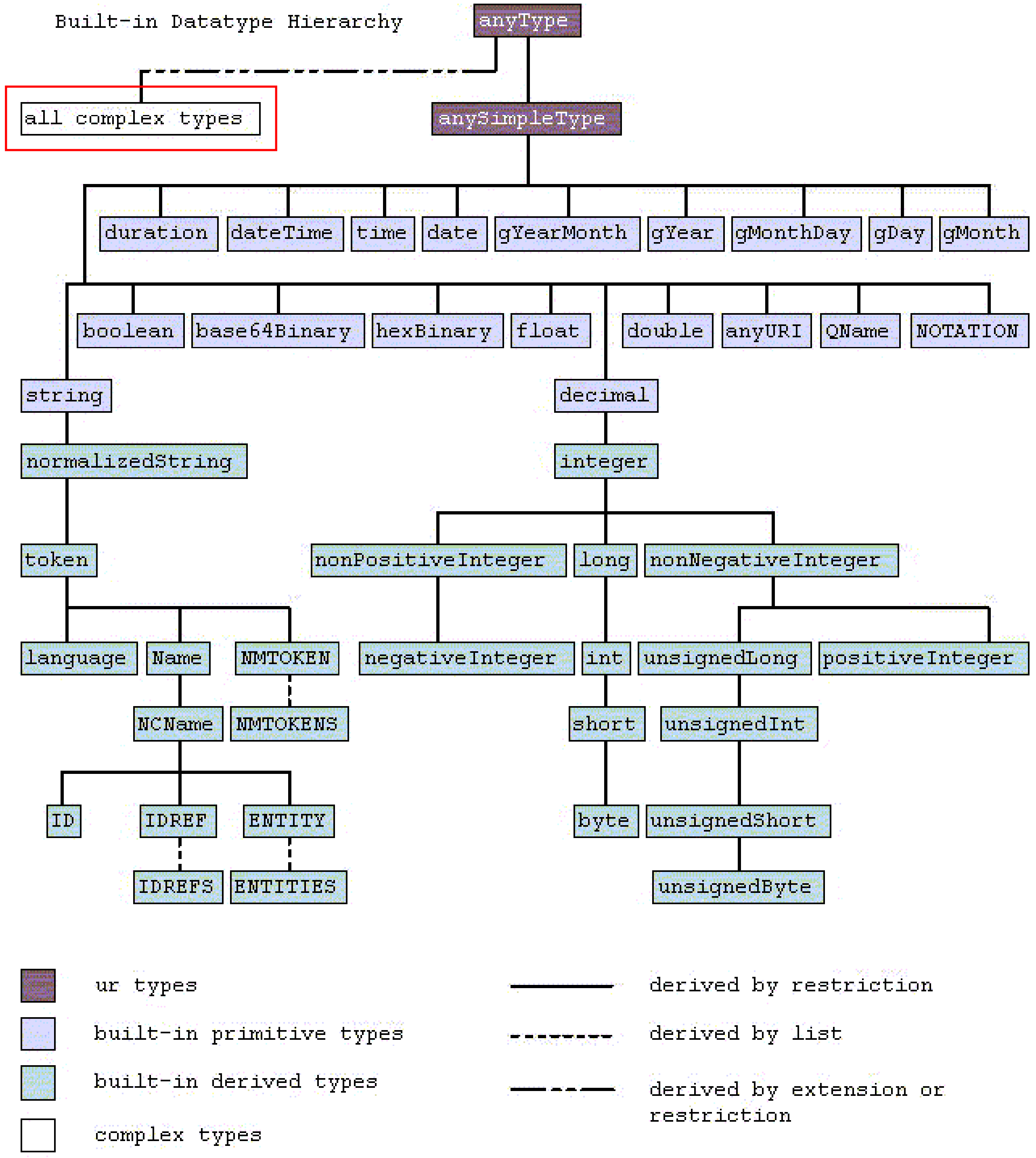
primitive

- they do not require defining derived
- derived from other types

derived

- derived form other types

W3C Recommendation



Si sfrutta la keyword complexType, ma non hai capito bene MA DOPO C'E' UN ESEMPIO.

## Complex types

Only the type defined globally
and named can be used many times

```
~~<xsd:element name="Student">
   <xsd:complexType name="personType">
    ...
   </xsd:complexType>
</xsd:element>~~
```

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

   <xsd:element name="Teacher" type="personType" />
   <xsd:complexType name="personType">
    ...
   </xsd:complexType>
</xsd:schema>
```

**Anonymous type**

**(può essere anonimo solo un elemento)**

Devi prima definirlo nella root come è stato fatto qui con name. Qui abbiamo definito il type come " personType" lo possiamo utilizzare più volte.

**Named type**

## Complex types

```
<xsd:element name=“Student” type=“personType” />
<xsd:element name=“Teacher” type=“personType” />
<xsd:complexType name=“personType”>
   .....
      <xsd:element name=“Surname” type=“xsd:string” />
      <xsd:element ref=“Name” />
      <xsd:element name=“Phone” type=“xsd:long” />
   .....
</xsd:complexType>
<xsd:element name=“Name” type=“xsd:string” />
```

ESEMPIO ALLA PAGINA DOPO

# Complex types

Order indicators

- way of occurrence of subelements in the complex type
- sequence
- choice
- all

```
<xsd:element name="Student" type="personType" />
<xsd:element name="Teacher" type="personType" />
<xsd:complexType name="personType">
  .....
     <xsd:element name="Surname" type="xsd:string" />
     <xsd:element ref="Name" />
     <xsd:element name="Phone" type="xsd:long" />
  .....
</xsd:complexType>
<xsd:element name="Name" type="xsd:string" />
```

The sentece in the XML document must appear in the order they are declare in the schema file

## Order indicators

sequence

- strict order of sub-elements

```
<xsd:complexType name=“personType”>
  <xsd:sequence>
     <xsd:element name=“Surname” type=“xsd:string” />
     <xsd:element name=“Name” type=“xsd:string” />
     <xsd:element name=“Phone” type=“xsd:long” />
 </xsd:sequence>
</xsd:complexType>
```

- Equivalent in DTD: (non ci interessa ora)

```
<!ELEMENT xyz (Surname, Name, Phone)>
```

# Order indicators

choice (LO USI PER LE ALTERNATIVE)

- only one of the declared child elements

```
<xsd:complexType name="Dane">
  <xsd:choice>
    <xsd:element name="Student" type="xsd:string" />
     <xsd:element name="Assistant" type="xsd:string" />
     <xsd:element name="Professor" type="xsd:string" />
  </xsd:choice>
</xsd:complexType>
```

- Equivalent in DTD:

```
<!ELEMENT xyz (Student | Assistant | Professor)>
```

Combination of chiose and sequence

# Order indicators

## Nesting choice and sequence

```
<xsd:sequence>
   <xsd:choice>
      <xsd:element name=“a” type=“xsd:long” />
      <xsd:element name=“b” type=“xsd:string” />
   </xsd:choice>
   <xsd:sequence>
      <xsd:element name=“c” type=“xsd:int” />
      <xsd:element name=“d” type=“xsd:float” />
   </xsd:sequence>
</xsd:sequence>
```

wrong abcd

correct acd

correct bcd

◦wrong bc

```
<!ELEMENT xyz ((a | b), (c, d))>
```

Fa vari esempi con questo, dalla slide così non puoi capire più di tanto (per ascolatare L.4 01:08:00

## Order indicators

Repeating the sequence/choice

```
<xsd:sequence maxOccurs=“unbounded”>
  <xsd:choice minOccurs=“0” maxOccurs=“1”>
    <xsd:element name=“a” type=“xsd:long” />
    <xsd:element name=“b” type=“xsd:string” />
  </xsd:choice>
  <xsd:sequence minOccurs=“0” maxOccurs=“unbounded”>
    <xsd:element name=“c” type=“xsd:int” />
    <xsd:element name=“d” type=“xsd:float” />
  </xsd:sequence>
</xsd:sequence>
```

```
< !ELEMENT xyz ((a | b)?, (c, d)*)+>
```

# Order indicators all

- all elements in any order (devono apparire)

```
<xsd:all>
   <xsd:element name="Surname" type="xsd:string" />
   <xsd:element ref="Name" />
   <xsd:element name="Phone" type="xsd:long" />
</xsd:all>
```

- limitations all
  - maxOccurs = ”1”, minOccurs = ”0” or ”1”
  - all cannot be nested within sequence, choice, all
  - in all only elements
- increasing the computational complexity of validating parsers
- Equivalent in DTD:

```
????
```

Su questo argomento lei hai molte più slide su cui spiega (data oriented vs document oriented)

## Mixed content type

It supports all properties of the complex type:

- minOccurs, maxOccurs, sequence ...

```
<complexType name=“MyName2” mixed=“true”>
  <sequence>
    <element name=“FirstName” maxOccurs=“2” type=“string” />
    <element name=“LastName” type=“string” />
  </sequence>
</complexType>
```

```
<person>
  Definicja dla osoby
  <FirstName>Anna</FirstName> bez innych imion
  <FirstName>Empty</FirstName>
  <LastName>Kowalska</LastName> z mieszaną zawartością
</person>
```

# anyType

The elements by default are "any type" *anyType*

- The following declarations are equivalent

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

```
<xsd:element name="Data" />
```

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

  <xs:element name="books" />
  <xs:complexType name="bookType">
    <xs:sequence>
      <xs:element name="title" type="xs:string"/
      <xs:element name="author" type="xs:string"
      <xs:element name="year" type="xs:short"/>
      <xs:element name="price" type="xs:float"/>
    </xs:sequence>
    <xs:attribute name="category" type="xs:strin
  </xs:complexType>
  <xs:complexType name="booksType">
    <xs:sequence>
      <xs:element name="book" type="bookType" ma
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

```
<?xml version="1.0" encoding="utf-8"?>
<books>
    <author>
        <name>student's name</name>
        <surname>student's surname</surname>
    </author>
    <study kind="lecture" >
        <activities id="1">
            <topic>Hypertext and hypermedia</topic>
            <range>
                <component>Hypertext &amp; hypermedia</component>
                <component>HTML CSS</component>
                <component>XML</component>
                <component>XML Schema</component>
                <component>DTD</component>
                <component>XSLT</component>
                <component>FO</component>
            </range>
            <score>30</score>
        </activities>
    </study>
    <study kind="laboratory" obligatory="yes">
        <activities id="1">
            <topic>HTML + CSS</topic>
            <range>
                <component>structure of the page</component>
                <component>links</component>
```

Manca tyoe="BookType" nella pirmo elemento!
quindi diventa "anytype" e lo schema sotto è
come viene letto.
Nota che il file XML è valido con la
dichiarazione sotto, perchè abbiamo usato
appunto "anytype". L'unico vincolo è che il
nome del root sia "Books"
--> NON USARE MAI ANYTYPE

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

  <xs:element name="books" />

</xs:schema>
```

```
      <price>29.99</price>
    </book>
    <book category="web">
      <title>Learning XML</title>
      <author>Erik T. Ray</author>
      <year>2003</year>
      <price>39.95</price>
    </book>
  </books>
```

## Element any

The <any> element enables us to extend the XML document with elements not specified by the schema.
Puoi scrivere altro dopo il cognome.

```
<complexType name=“personType” mixed=“true”>
    <sequence>
        <element name=“FirstName” maxOccurs=“2” type=“string” />
        <element name=“LastName” type=“string” />
        <any minOccurs="0"/>
    </sequence>
</complexType>
```

## Empty content

element may be empty

Dichiaro un elemento complesso ma dentro non ci metto nulla

```
<xsd:element name="emptyElement">
   <xsd:complexType />
</xsd:element>
```

```
<emptyElement />
```

Poi c'è una slide che non ha messo qui

## Element group

the group element is used to define a group of elements to be used in complex type definitions.

named groups of model, fragments of content models that you can reuse

benefits

- indicates that some complex types have similar subelements
- allows you to create more compact schema

**The group can not contain both elements and attributes**

Groups must be declared globally

Esempio of group element:

```
<complexType name="StudentType" mixed="true">
   <sequence>
      <group ref="MyGroup" />
   </sequence>
</complexType>

<complexType name="TeacherType" mixed="true">
   <sequence>
      <element name="Title" maxOccurs="3" type="string" />
      <group ref="MyGroup" />
   </sequence>
</complexType>

<group name="MyGroup" >
   <sequence>
      <element name="FirstName" type="string" />
      <element name="LastName" type="string" />
   </sequence>
</group>
```

## Simple types

Creating your **own** simple datatypes

Simple types inherit (ereditare) from:

- built-in types
- other simple types

Only simple content (not sub element or attributes are allowed in simple types)

# General form of creating a new datatype ( SIMPLETYPE) by specifying facet values

ESEMPIO A PAGINA DOPO

```
<simpleType name="name" >
    <restriction  base="source">
        <facet value="..." />
        <facet value="..." />
        ...
    </restriction>
</simpleType>
```

Facets:
- length
- minlength
- maxlength
- pattern
- enumeration
- minInclusive
- maxInclusive
- minExclusive
- maxExclusive

...

Sources:
- string
- boolean
- number
- float
- double
- duration
- dateTime
- time

...

## Example of creating a new datatype by specifying facet values

```
<xsd:simpleType name="TelephoneNumberType">
    <xsd:restriction base="xsd:string">
        <xsd:length value="8"/>
        <xsd:pattern value="\d{3}-\d{4}"/>
    </xsd:restriction>
</xsd:simpleType>
```

Deve essere lunga 8 caratteri

Deve poi seguire la regola:
3numeri - 4 numeri

## Facets of the string

The string primitive datatype has six optional facets:

- length
- minLength
- maxLength
- pattern
- enumeration
- whitespace (legal values: preserve, replace, collapse)

```
<simpleType name="AustrianZIPCode" >
   <restriction  base="string">
      <length value="4" />      La lunghezza sarà 4
   </restriction>
</simpleType>
<simpleType name="InternationalZIPCode" >
   <restriction  base="string">
      <minLength value="4" />     la lunghezza sarà
      <maxLength value="6" />     compresa fra 4 e 6
   </restriction>
</simpleType>
```

## enumeration

- List of predefined values (per forzare il valore fra alcuni scelti)

```
<day> Tuesday </day>
```

```
<simpleType name="dayType" >
   <restriction base="string">
      <enumeration value="Monday" />
      <enumeration value="Tuesday" />
      <enumeration value="Wednesday" />
      <enumeration value="Thursday" />
      <enumeration value="Friday" />
   </restriction>
</simpleType>
```

# pattern

The value of pattern  must be a regular expression
Si definisce un set di valori validi

[0-9] --> range di valori
{3} --> quanti valori devi mettere
esempio: [0-9]{2} --> 2 valori fra 0 e 9

```
<xsd:simpleType name="typNIP">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="([0-9]{3}-[0-9]{2}-[0-9]{2}-[0-9]{3})|
                    ([0-9]{3}-[0-9]{3}-[0-9]{2}-[0-9]{2})"/>
  </xsd:restriction>
</xsd:simpleType>
```

111-22-33-444-555-666-77-88

# Regular expressions

| | |
|---|---|
| \p{L} | A letter, from any language |
| \p{Lu} | An uppercase letter, from any language |
| \p{Ll} | A lowercase letter, from any language |
| \p{N} | A number - Roman, fractions, etc |
| \p{Nd} | A digit from any language |
| \p{P} | A punctuation symbol |

? --> means 0 or 1 times
* --> means 0 or many times
+ --> means 1 or many times

```
<xsd:simpleType name="money">
   <xsd:restriction base="xsd:string">
      <xsd:pattern value="\p{Sc}\p{Nd}+(\.\p{Nd}\p{Nd})?"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:element name="cost" type="money"/>
```

Currency sign

Digit from any language

```
<cost>$45.99</cost>
<cost>¥300</cost>
```

Ci sono molti altri esempio durante la lezione

# Facets of the integer datatype

The integer datatype has 8 optional facets:

totalDigits

pattern

whitespace

enumeration

maxInclusive

maxExclusive

minInclusive

minExclusive

```
<simpleType name="gradeType" >
   <restriction base="Integer">
      <minInclusive value="1" />
      <maxInclusive value="5" />
   </restriction>
</simpleType>
```

I valori possono andare da 1 a 5

## Facets of the decimal datatype

The decimal datatype has 9 optional facets:

- totalDigits
- fractionDigits
- pattern
- whitespace
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

```
<simpleType name="4digitnumberType" >
   <restriction base="decimal">
      <totalDigits value="4" />
   </restriction>
</simpleType>
```

Massimo numero di cifre in generale(?)

```
<simpleType name="4plus2digitnumberType" >
   <restriction base="decimal">
      <totalDigits value="6" />
      <fractionDigits value="2" />
   </restriction>
</simpleType>
```

Massimo numero di cifre dopo la virgola

# Multiple Facets - "*and*" them together, or "*or*" them together?

```
<xsd:simpleType name="TelephoneNumberType">
   <xsd:restriction base="xsd:string">
      <xsd:length value="8"/>
      <xsd:pattern value="\d{3}-\d{4}"/>
   </xsd:restriction>
</xsd:simpleType>
```

An element declared to be of type TelephoneNumber must be a string of length=8 *and* the string must follow the pattern: 3 digits, dash, 4 digits.

```
<xsd:simpleType name="shapeType">
   <xsd:restriction base="xsd:string">
     <xsd:enumeration value="circle"/>
     <xsd:enumeration value="triangle"/>
     <xsd:enumeration value="square"/>
   </xsd:restriction>
</xsd:simpleType>
```

An element declared to be of type shape must be a string with a value of *either* circle, *or* triangle, *or* square.

**Patterns, enumerations => "or" them together All other facets => "and" them together**

## Fixing a facet value

```
<xsd:simpleType name= "daysType">
   <xsd:restriction base="xsd:int">
      <xsd:minInclusive value="1" fixed="true"/>
      <xsd:maxInclusive value="30"/>
   </xsd:restriction>
</xsd:simpleType>
```

Provi a cambiare valore in un elemento derivato , ma non puoi cambiare il valore nell' elemento padre è fixed="true"

```
<xsd:simpleType name= "myDaysType">
   <xsd:restriction base=" daysType ">
      <xsd:minInclusive value="7"/>
      <xsd:maxInclusive value="30"/>
   </xsd:restriction>
</xsd:simpleType>
```

## Summary (riassunto) of declaring elements

```
<xsd:element name="name" type="type" minOccurs="int„ maxOccurs="int"/>
```

```
<xsd:element name="name" minOccurs="int" maxOccurs="int">
    <xsd:complexType>
        …
    </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="name" minOccurs="int" maxOccurs="int">
    <xsd:simpleType>
        <xsd:restriction base="type">
        …
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
```

# Lists

```
<numbers>2 10 6 1 15 150</numbers>
```

The list element defines a simple type element as a list of values of a specified data type.

The list can only contain simple types

You cannot create a list of lists

All list items of the same type

List elements in XML documents must be separated by whitespace

facets allowed on list

- *length, minLength, maxLength*
  - they determine the length of the list
- *enumeration, pattern*
  - they specify the values of t

```
<xsd:simpleType name="numbersType">
  <xsd:list itemType=" xsd:positiveInteger "/>
</xsd:simpleType>

<xsd:element name="numbers" type="numbersType "/>
```

PUoi unire ogni tipo di file, ma solo simple type. Qui possiamo vedere due versioni della stessa cosa Ma nel secondo modo i simpletype non sono riutilizzabili perchè sono all'interno.

## Union

Connecting any simple types into one

You can combine different types

```
<class>1</class>
<class>One</class>
```

```
<simpleType name="Type1" >
   <restriction base="string">
      <enumeration value="One" />
      <enumeration value="Two" />
   </restriction>
</simpleType>
<simpleType name="Type2" >
   <restriction base="positiveInteger">
     <maxInclusive value="2"/>
  </restriction>
</simpleType>
<simpleType name="unionType" >
   <union memberTypes="Type1 Type2">
</simpleType>
<element name="class" type="unionType" />
```

```
<simpleType name="unionType" >
   <union>
      <simpleType>
         <restriction base="string">
            <enumeration value="One" /> ...
         </restriction>
      </simpleType>
      <simpleType>
       <restriction base="positiveInteger">
           <maxInclusive value="2"/>
          </restriction>
      </simpleType>
   </union>
</simpleType>
```

Ci sono due vie: restringere un type oppure aggiungere valori a un base type

## Derived complex types

restriction

- the set of values of the new type is a subset
- similar to simple types

extension

- adding additional elements to the base type

Base attributes indicates the base type

## Derive by extension

```
<complexType name="personType" >
  <sequence>
    <element name="name" type="string" />
    <element name="surname" type="string" />
  </sequence>
</complexType>
```

```
<complexType name="extendedType" >
  <complexContent>
    <extension base="personType">
      <sequence>
        <element name="email" type="string" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Protremmo volere che da un elemento non si possa derivare nulla

## Derive by extension

```
<complexType name="personType"   final="#all"  >
   <sequence>
      <element name="name" type="string" />
      <element name="surname" type="string" />
   </sequence>
</complexType>
```

```
<complexType name="extendedType" >
   <complexContent>
      <extension base="personType">
         <sequence>
            <element name="email" type="string" />
         </sequence>
      </extension>
   </complexContent>
</complexType>
```

dARA' ERRORE!!!

Element of the restricted type must be a valid element of the base type.
NOTA: Se vuoi eliminare un elemento nell'elemento derivato, esso deve necessariamente essere opzionale nel base type

# Derive by restriction

```
<xsd:complexType name="Publication">
        <xsd:sequence>
                <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
                <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
                <xsd:element name="Date" type="xsd:gYear"/>
        </xsd:sequence>
</xsd:complexType>
<xsd:complexType name= "SingleAuthorPublication">
    <xsd:complexContent>
        <xsd:restriction base="Publication">
            <xsd:sequence>
                <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
                <xsd:element name="Author" type="xsd:string"/>
                <xsd:element name="Date" type="xsd:gYear"/>
            </xsd:sequence>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
```

# Attributes

The attribute declarations always come last, after the element declarations.
(le dichiarazione di attributi viene sempre dopo quella degli elementi)

Declared localy or globaly

Always simple type

```
<attribute name=“attr1” type=“string” />
<element name=“testelement”>
   <complexType>
      <sequence>
      ...
      </sequence>
      <attribute ref=“attr1” />
      <attribute name=“attr2” type=“string” />
   </complexType>
</element>
```

Global attribute

reference

Local attribute

## Attributes

default = automaticamente aggiunto all'attributo

fixed = automenticamente aggiunto all'attributo, ma non si può cambiare

Questi attributi hanno senso solo all'interno di elementi non negli attributi globali

use= ”optional”

use= ”required”

use= ”prohibited” Si capisce l'attributo di questo dal video (si usa per gli elementi derivate (quelli ristretti))

```
<xs:attribute name="lang" type="xs:string" use="required"  fixed="EN"/>
```

## anyAttribute

Permette di aggiungere attributi che non sono specificati dallo schema

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:any minOccurs= ”0” />
    </xs:sequence>
    <xs:anyAttribute/>
  </xs:complexType>
</xs:element>
```

E' usato per raggruppare un gruppo di attributi

## attributeGroup

```
<element name=“testelement2”>
   ...
   <attributeGroup ref=“MyAttrGroup1” />
   ...
</element>
<attributeGroup name=“MyAttrGroup1”>
   <attribute name=“attr1” type=“long” />
   <attribute name=“attr2” >
      <simpleType> ... </simpleType>
   </attribute>
</attributeGroup>
```

Ci sono due modi:

## Summary of Declaring Attributes

```
<xsd:attribute name="name" type="simple-type" use="how-its-used" default/fixed="value"/>
```

xsd:string
xsd:integer
xsd:boolean
...

required
optional
prohibited

The "use" attribute must be optional if you use default or fixed.

```
<xsd:attribute name="name" use="how-its-used" default/fixed="value">
    <xsd:simpleType>
        <xsd:restriction base="simple-type">
            <xsd:facet value="value"/>
            ...
        </xsd:restriction>
    </xsd:simpleType>
</xsd:attribute>
```

C' E' UNA SLIDE MOLTO IMPORTANTE SUL VIDEO, COPIALA!!!

NON HAI CAPITO MOLTO BENE...

```
<element name=”size”>
<complexType >
    <simpleContent>
        <extension base=“integer” >
            <attribute name=“system” type=“token” />
        </extension>
     </simpleContent>
</complexType>
</element>
```

```
<size system=”US-DRESS”>10</size>
```

```
<element name=”size”>
<complexType>
       <attribute name=“system” type=“token” />
</complexType>
</element>
```

```
<size system=”US-DRESS”/>
```

Include and import are used to add multiple schemas to a document

## Include and import

include

```
<schema xmlns=“http://www.w3.org/2001/XMLSchema
        targetNamespace=“http://www.example.org/NS2”>
<include schemaLocation=“SchemaToInclude.xsd” />
</schema>
```

import

```
<schema xmlns=“http://www.w3.org/2001/XMLSchema
   targetNamespace=“http://www.example.org/NS2”
   xmlns:imp1=“http://www.importme.org/Import1”>
<import namespace=“http//www.importme.org/Import1
                    /schemas/Import1.xsd” />
<element name=“test1” type=“imp1:testType” />
</schema>
```

## Annotations

for people <documentation>

for applications <appinfo>

Nulla a effetto sull Schema validation. Ma non puoi metterle ovunque: solo prima e dopo ogni elemento globale (ma alla fine si può lo stesso..) --> la sclide che lo spiega qui non c'è)

```
<xsd:annotation>
   <xsd:documentation>
      This text is intended for humans.
   </xsd:documentation>
   <xsd:appinfo>
      <someXML>any well-formed XML</someXML>
   </xsd:appinfo>
</xsd:annotation>
```

# nil (annullabile) content

attribute nillable

**indicates that the element may not have content**

in XML xsd:nill=”true”

```
<complexType name=“Price”>
   <sequence>
      <element name=“amount” type=“integer” />
      <element name=“currency” type=“string” nillable=“true” />
   </sequence>
</complexType>
```

```
<Price>
   <amount>100</amount>
   <currency xsd:nil=“true” />
</Price>
```

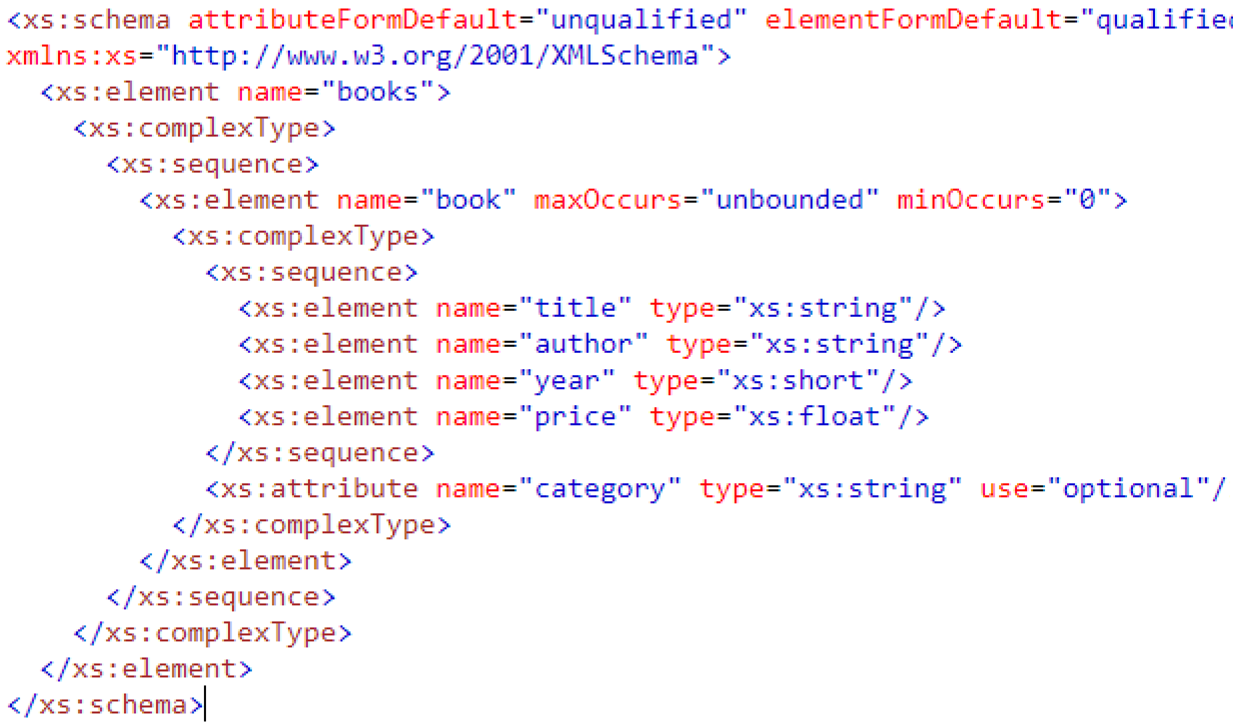
# XML Schema design methods

We can use 3 strategies:

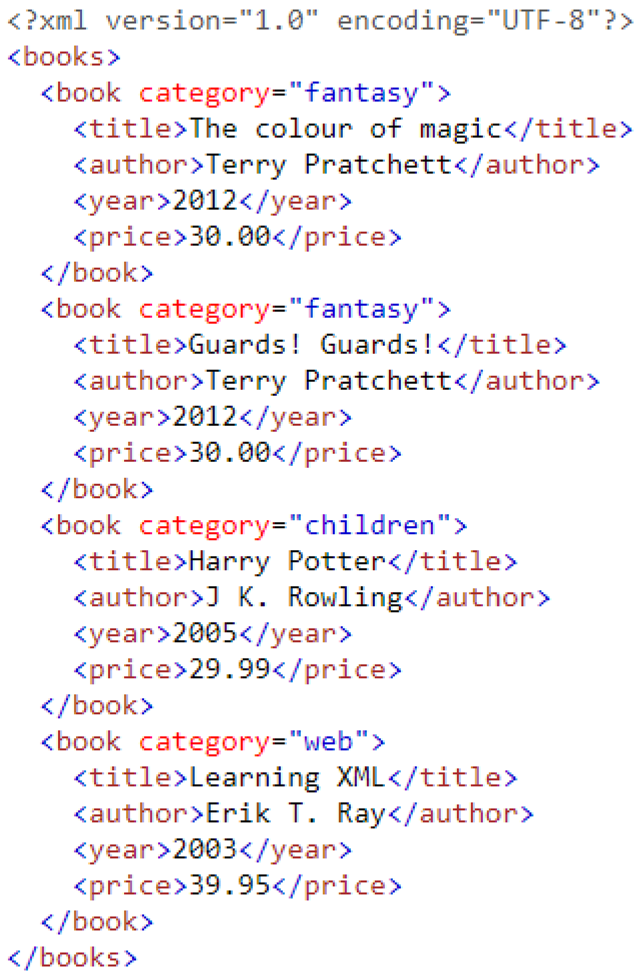
Define the type of each element using a local type

Define a series of named complex and simple types at the top level of the XML Schema document and use those names to indicate the types to be used for the elements

Define a series of elements and groups of code at the top level of the Schema definition and then refer to those element definitions using the attribute ref



```xml
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="books">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="book" maxOccurs="unbounded" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="title" type="xs:string"/>
              <xs:element name="author" type="xs:string"/>
              <xs:element name="year" type="xs:short"/>
              <xs:element name="price" type="xs:float"/>
            </xs:sequence>
            <xs:attribute name="category" type="xs:string" use="optional"/
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```



```xml
<?xml version="1.0" encoding="UTF-8"?>
<books>
  <book category="fantasy">
    <title>The colour of magic</title>
    <author>Terry Pratchett</author>
    <year>2012</year>
    <price>30.00</price>
  </book>
  <book category="fantasy">
    <title>Guards! Guards!</title>
    <author>Terry Pratchett</author>
    <year>2012</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</books>
```

hCI sono varie possiblità, qui è definito tutto localmente

```xml
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="books" type="booksType"/>
  <xs:complexType name="bookType">
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="author" type="xs:string"/>
      <xs:element name="year" type="xs:short"/>
      <xs:element name="price" type="xs:float"/>
    </xs:sequence>
    <xs:attribute name="category" type="xs:string" use="optional"/>
  </xs:complexType>
  <xs:complexType name="booksType">
    <xs:sequence>
      <xs:element name="book" type="bookType" maxOccurs="unbounded" minOcc
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<books>
  <book category="fantasy">
    <title>The colour of magic</title>
    <author>Terry Pratchett</author>
    <year>2012</year>
    <price>30.00</price>
  </book>
  <book category="fantasy">
    <title>Guards! Guards!</title>
    <author>Terry Pratchett</author>
    <year>2012</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</books>
```

Altra strategia...

```xml
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="title" type="xs:string"/>
  <xs:element name="author" type="xs:string"/>
  <xs:element name="year" type="xs:short"/>
  <xs:element name="price" type="xs:float"/>
  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="title"/>
        <xs:element ref="author"/>
        <xs:element ref="year"/>
        <xs:element ref="price"/>
      </xs:sequence>
      <xs:attribute name="category" type="xs:string" use="optional"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="books">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="book" maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<books>
  <book category="fantasy">
    <title>The colour of magic</title>
    <author>Terry Pratchett</author>
    <year>2012</year>
    <price>30.00</price>
  </book>
  <book category="fantasy">
    <title>Guards! Guards!</title>
    <author>Terry Pratchett</author>
    <year>2012</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</books>
```

Altra strategia