XML Schema

Arantza Irastorza Goñi

eman ta zabal zazu

Informazioaren Kudeaketa Aurreratua

Gradua Ingeniaritza Informatikoan

Esp. Software Ingeniaritza

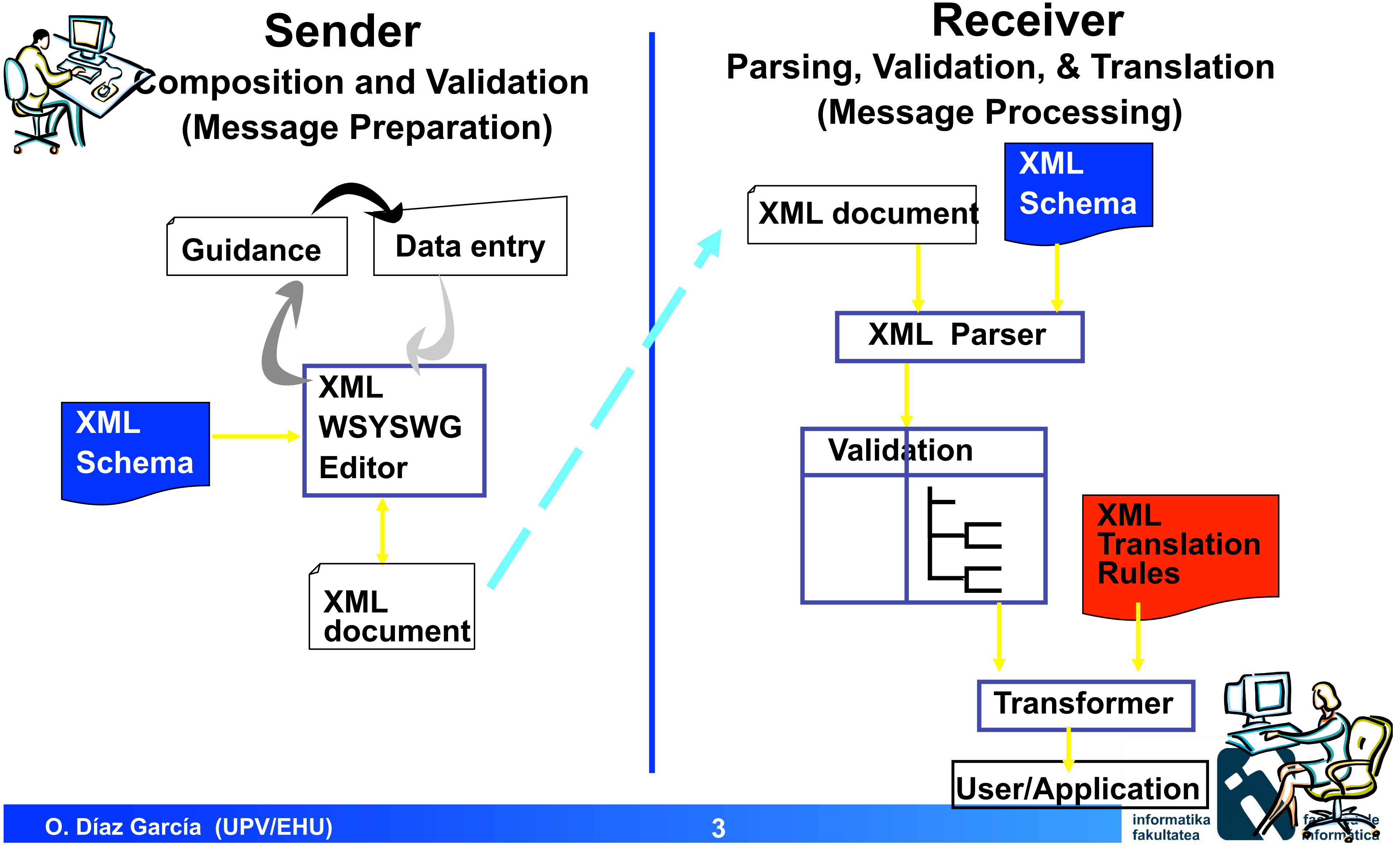
Lengoaiak eta Sistema Informatikoak saila

informatika fakultatea

facultad de informática

2017 urtarrila

The big picture!



informatika fakultatea facultad de informática

Contents

© Acknowledgments. Part of these slides have been prepared using R.L. Costello's material

Motivation: Is this document valid?

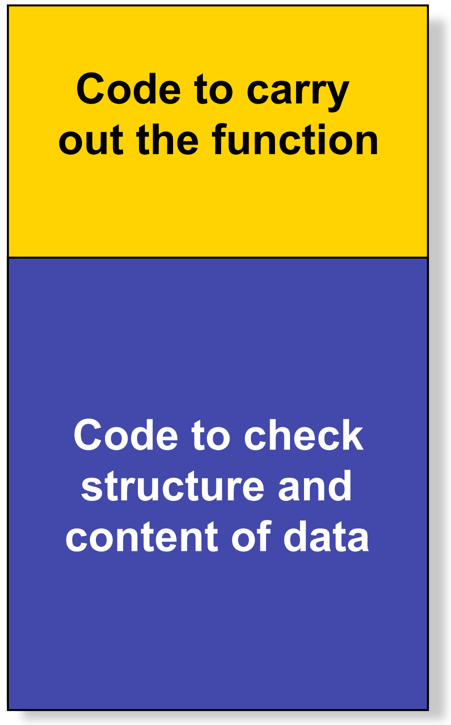
```
<location>
        <latitude>32.904237</latitude>
        <longitude>73.620290</longitude>
        <uncertainty unit="meter">2</uncertainty>
</location>
```

To be valid it must satisfy the following conditions:

1. Location is composed of latitude followed by longitude and then an indication of the precision of these measures
2. Latitude must be a decimal between -90 and +90
3. Longitude must be a decimal between -180 and +180
4. Both latitude and longitude must have exactly six numbers after the decimal point
5. The precision value must be positive
6. Precision is measured in meters or feet

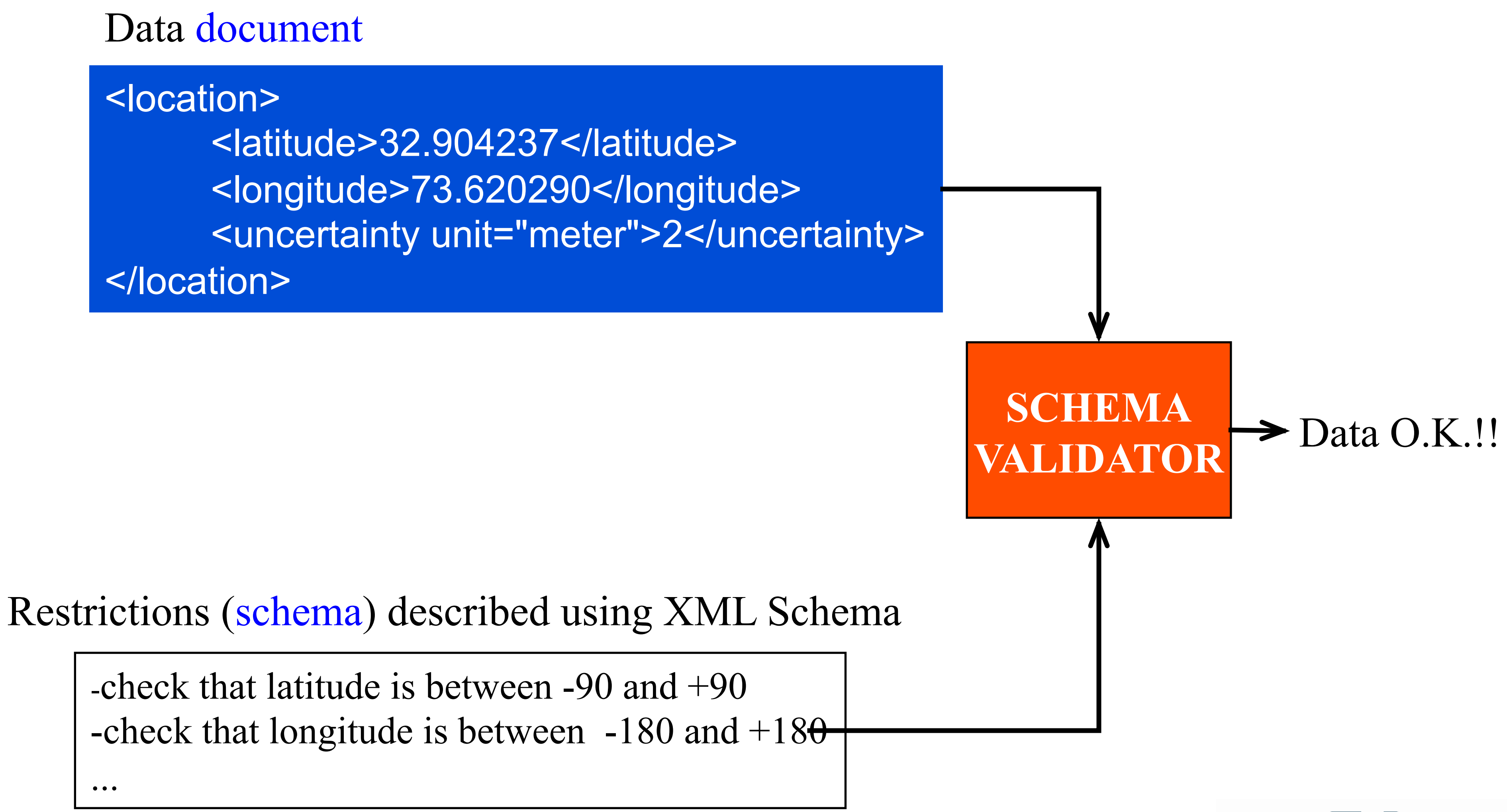
All these restrictions can be expressed with XML Schema

Who validates?
The receiving application


Code to carry out the function
Code to check structure and content of data

On average, up to **60%** of the code is used to verify the data

Who validates?
The validator, a general-purpose application



Data document

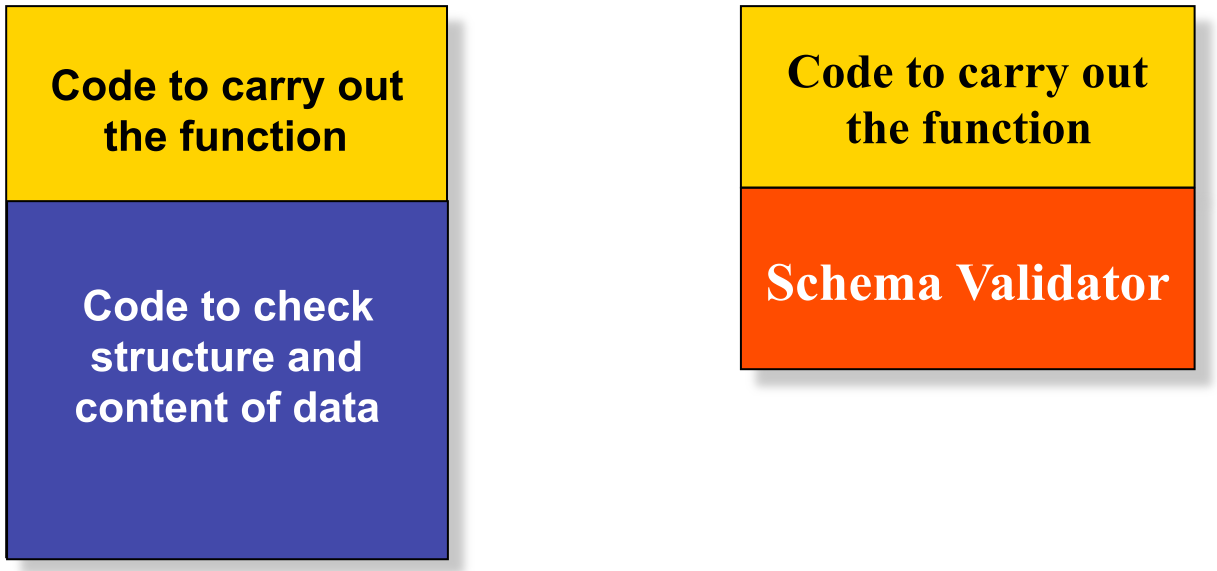
```
<location>
        <latitude>32.904237</latitude>
        <longitude>73.620290</longitude>
        <uncertainty unit="meter">2</uncertainty>
</location>
```

Restrictions (schema) described using XML Schema

-check that latitude is between -90 and +90
-check that longitude is between -180 and +180
...

Advantage

- Code is reduced and therefore, development and maintenance cost


Code to carry out the function
Code to check structure and content of data
Code to carry out the function
Schema Validator

Some terminology

- Validation
  - process that checks if an XML document follows the rules stated by a given schema
- Validator
  - a general purpose program for conducting validations from a declarative schema
- "Well formed" document
  - one that follows XML rules
- "Valid" document
  - one that follows rules of a <u>schema</u> (if there is one)

informatika fakultatea facultad de informática

Schema determines

- What sort of elements can appear in the document
- What elements MUST appear
- Which elements can appear as part of another element
- What attributes can appear or must appear
- What kind of values can/must be in an attribute

What is *XMLSchema?*

- W3C standard to define schemas (recommendation since May 2001)

http://www.w3.org/2001/XMLSchema

- This namespace allows specifying a schema, i.e.:
  - structure of data
  - type of each element/attribute

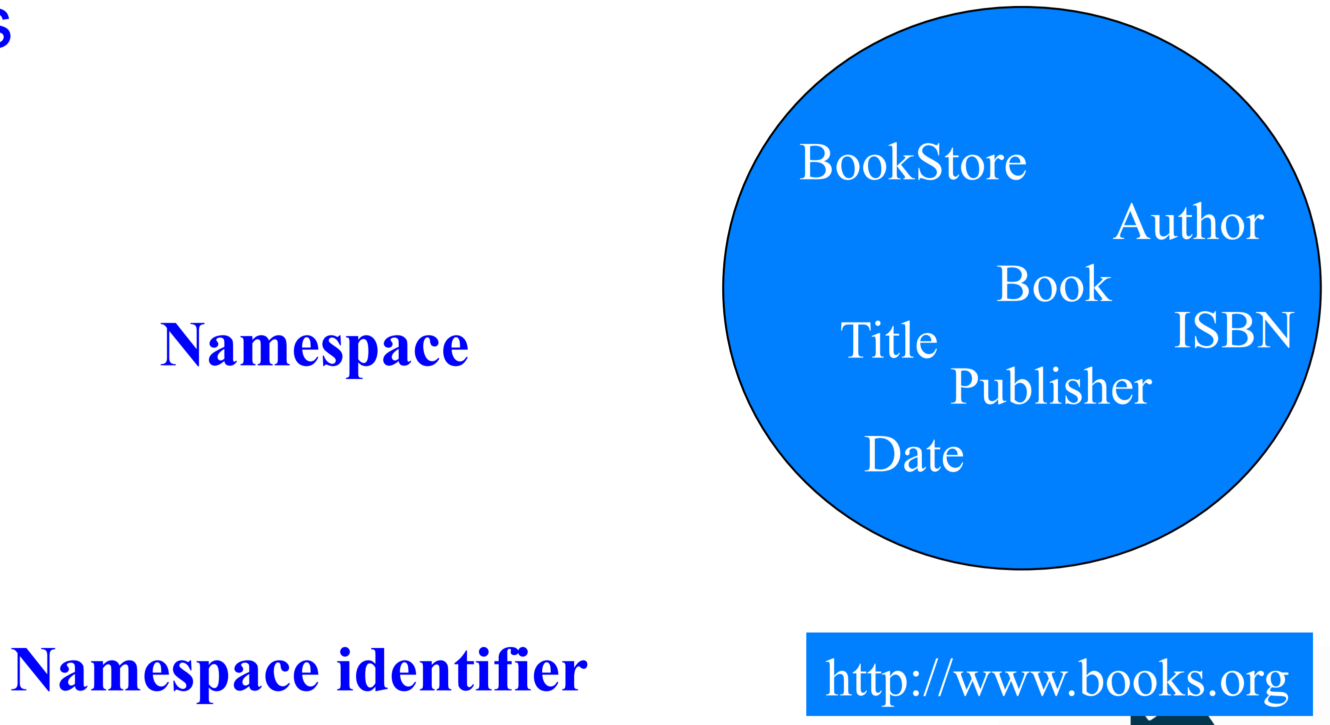
- The schema is also an XML document

Contents

© Acknowledgments. Part of these slides have been prepared using R.L. Costello's material

Schema = identifier + vocabulary

- Schema or vocabulary or namespace
  - for expressing the business rules of one's data
- This vocabulary is used to describe "instance" documents



Schema = <u>**identifier**</u> + vocabulary

- A namespace is only a string… unique in the whole wide world

- Syntax: Uniform Resource Identifier (URI) Norm

- Two options
  - Uniform Resource Locator (URL)
    - http://www.onekin.org/myBooks
  - Uniform Resource Names (URN)
    - urn:www-onekin-org:myBooks

Identifier: uniqueness

- A namespace must guarantee that each term defined in the space is unique

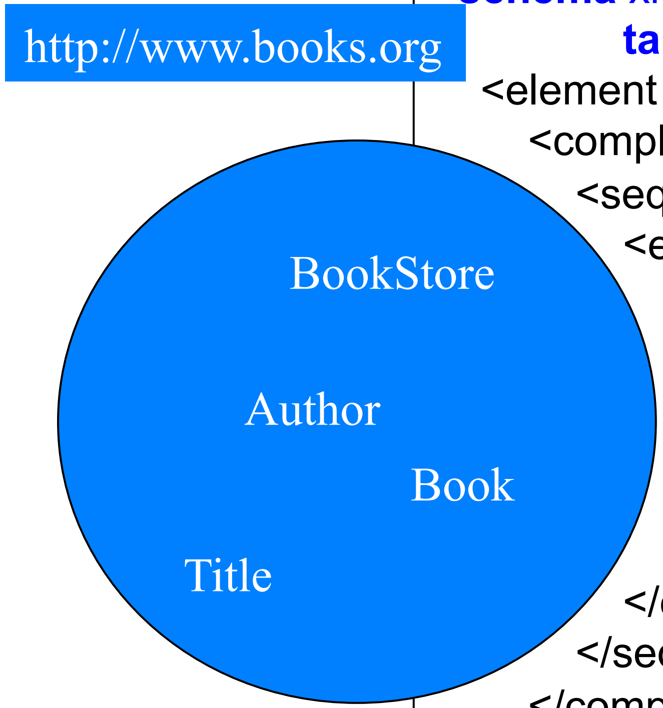
- But, to single out each "term" in the "whole wide world", the namespace identifier must be unique too

- Guarantee: *"Internet Naming Authority"*

$$\underbrace{\textit{http://www.onekin.org}}_{\text{Uniqueness guaranteed by the "Internet Naming Authority"}}\underbrace{\textit{/myBooks/science}}_{\text{Uniqueness guaranteed by yourself}}$$

Schema = identifier + vocabulary

A schema defines the vocabulary and restrictions that control the creation of new “document instances”



```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.books.org“ …>
  <element name="BookStore">
    <complexType>
      <sequence>
        <element name="Book"  maxOccurs="unbounded">
          <complexType>
            <sequence>
              <element name="Title" type="xs:string"/>
              <element name="Author" type="xs:string"/>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>
```

Contents

© Acknowledgments. Part of these slides have been prepared using R.L. Costello’s material

tad de
nática

Schema is a XML document

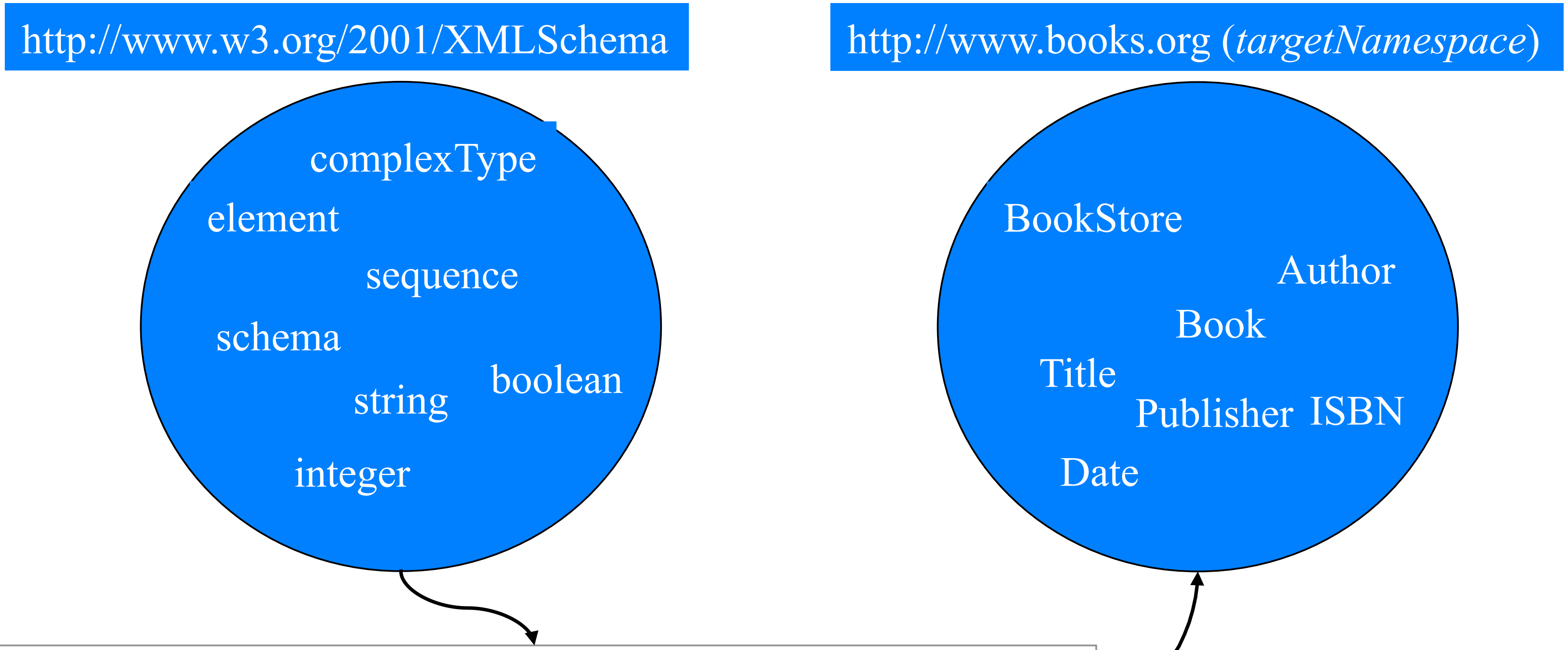
- Extension: ".xsd"

- Being an XML document,
  - the XML rules must be followed
  - an XML editor can be used to write them
  - DOM can be used to manipulate them
  - XSLT can be used to transform them

Its structure/vocabulary is dictated by a (meta) schema defined by W3C

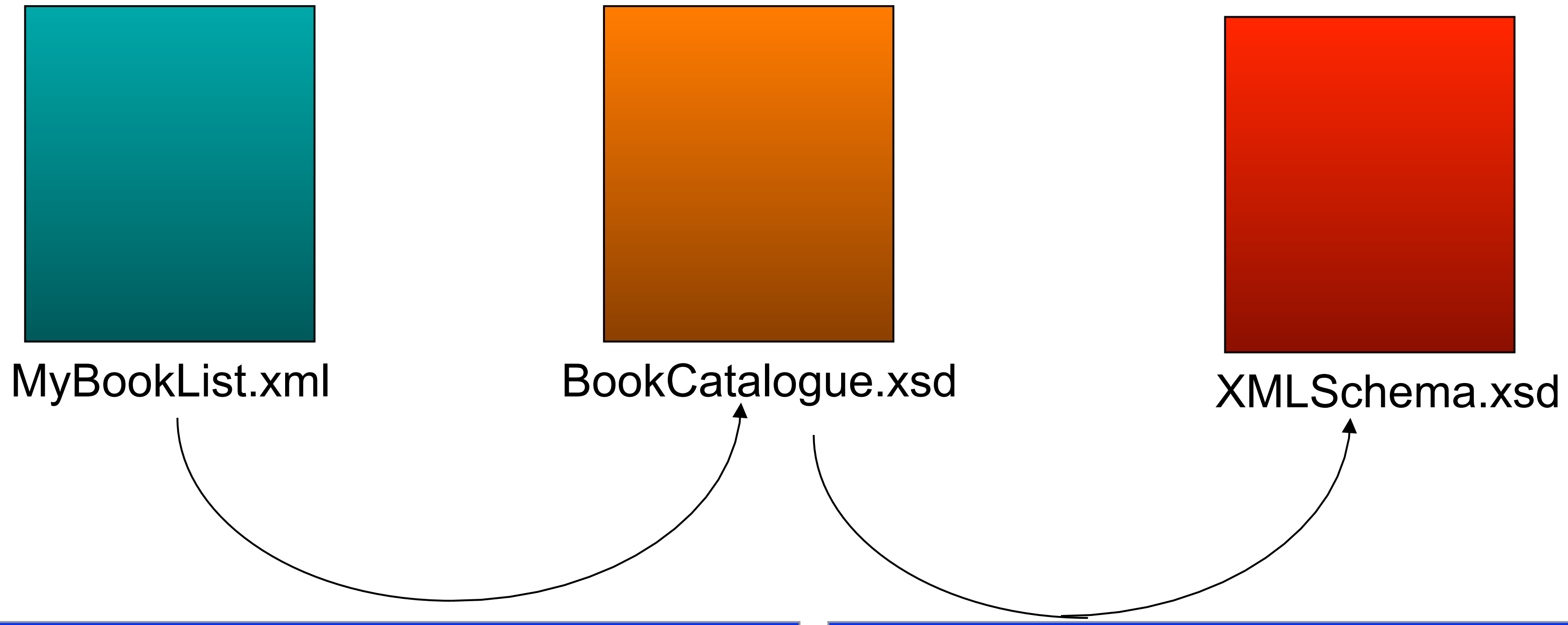
The schema as an XML document

A schema is defined using a (meta) schema: *XML Schema*



This is the XML Schema vocabulary that allows you to define YOUR OWN schema to describe books

The schema as an XML document (2)



Is this XML document **valid** according to the rules defined in **BookCatalogue.xsd**?

Is this document **valid** according to the rules defined in en **XMLSchema.xsd**?

The schema as an XML document (3)

```
<? xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.books.org" …>
  <element name="BookStore">
    <complexType>
      <sequence>
        <element name="Book"  maxOccurs="unbounded">
          <complexType>
            <sequence>
              <element name="Title" type="string"/>
              <element name="Author" type="string"/>
              <element name="Date" type="string"/>
              <element name="ISBN" type="string"/>
              <element name="Publisher" type="string"/>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>
```

BookCatalogue.xsd

The root is always a *schema* element

The schema is an XML document, therefore it has an associated schema

URL where the vocabulary being defined will be left

Root element of "book" documents

Association a schema to a document. Schema location

- Aim: providing hints for the validator to locate the schema

- If no location is provided it is up to the validator to find the schema on its own

```
<? xml version="1.0"?>
<BookStore xmlns="http://www.books.org"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.books.org
                               BookCatalogue.xsd”>
  <Book>
        <Title>My Life and Times</Title>
        <Author>Paul McCartney</Author>
        <Date>1998</Date>
        <ISBN>1-56592-235-2</ISBN>
        <Publisher>McMillin Publishing</Publisher>
  </Book>
  <Book>
        <Title>Illusions: The Adventures of a Reluctant Messiah</Title>
        <Author>Richard Bach</Author>
        <Date>1977</Date>
        <ISBN>0-440-34319-4</ISBN>
        <Publisher>Dell Publishing Co.</Publisher>
   </Book>
</BookStore>
```

MyBookList.xml

http://www.w3.org/TR/xmlschema-1/#schema-loc

Association a schema to a document: Example

note.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.w3schools.com"
    xmlns="http://www.w3schools.com"
    elementFormDefault="qualified">
<xs:element name="note">
    <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>
```

adib.xml

```
<?xml version="1.0"?>
<note
    xmlns="http://www.w3schools.com"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3schools.com note.xsd">

    <to>Jon</to>
    <from>Mikel</from>
    <heading>Gogoratu</heading>
    <body>Astebukaera honetan Bilbora goaz</body>
</note>
```

© http://www.w3schools.com/schema/schema_howto.asp

Association a schema to a document. Option 1

```
<?xml version="1.0"?>
<BookSeller xmlns="http://www.BookRetailers.org"/>
<Book>
        <Title>My Life and Times</Title>
        <Author>Paul McCartney</Author>
        <Date>1998</Date>
        <ISBN>1-56592-235-2</ISBN>
        <Publisher>McMillin Publishing</Publisher>
        <Reviewer xmlns="http://www.books.org" cod="12">
          <name>
            <First>Roger</First>
            <Last>Costello</Last>
          </name>
        </Reviewer>
    </Book>
    <Book>
        <Title>Illusions: The Adventures of a Reluctant Messiah</Title>
        <Author>Richard Bach</Author>
        <Date>1977</Date>
        <ISBN>0-440-34319-4</ISBN>
        <Publisher>Dell Publishing Co.</Publisher>
    </Book>
</BookSeller>
```

Using the *xmlns* attribute
It indicates the vocabulary in which the element is defined

The attribute does not "inherit" the new vocabulary !!!

The subelements with no xmlns attribute "inherit" it from the element that contains them

Association a schema to a document. Option 2

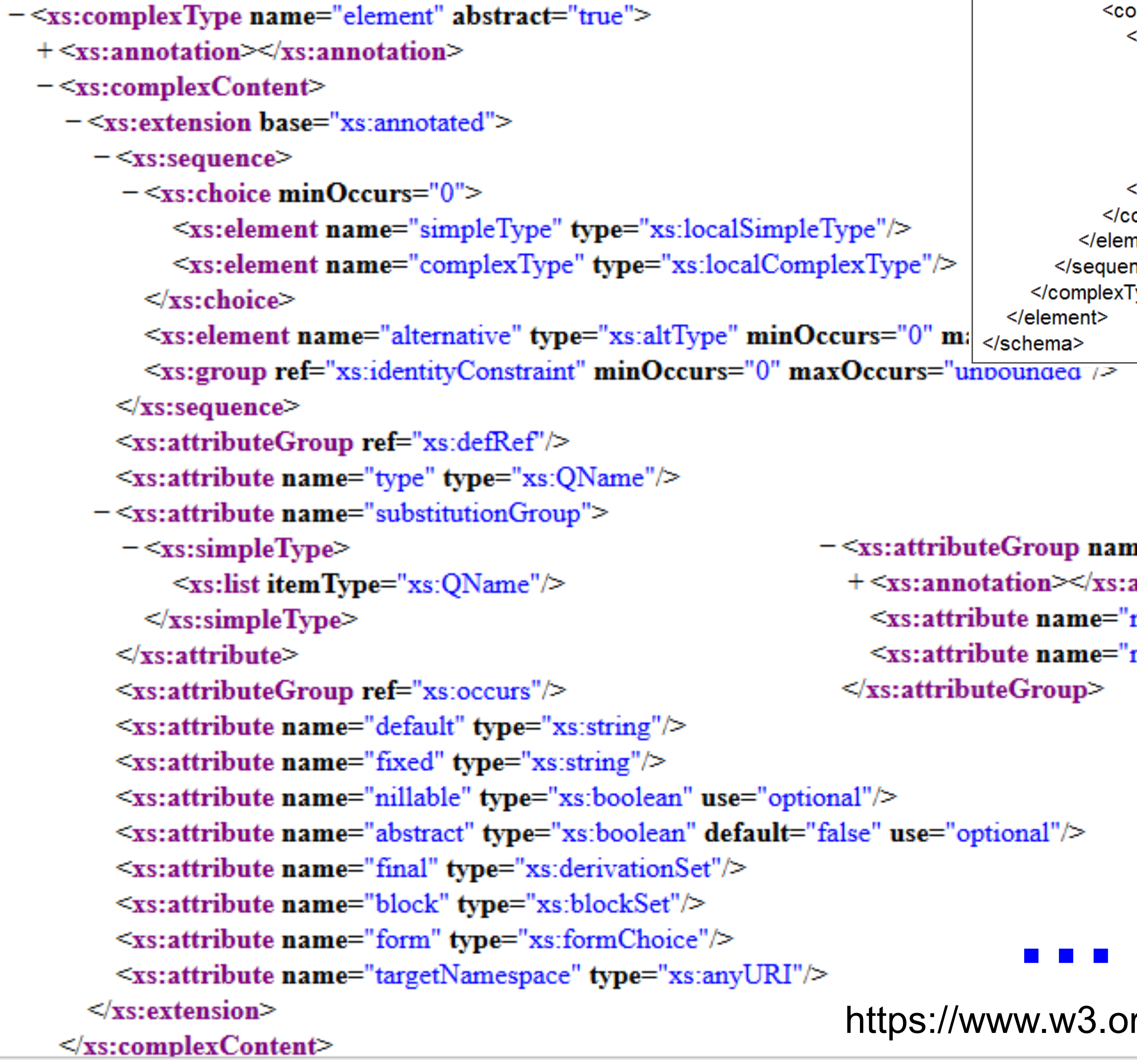
```
<?xml version="1.0"?>
<BookSeller xmlns="http://www.BookRetailers.org"
          xmlns:bns="http://www.books.org“ />
<Book bns:id="P.M.">
          <Title>My Life and Times</Title>
          <Author>Paul McCartney</Author>
          <Date>1998</Date>
          <ISBN>1-56592-235-2</ISBN>
          <Publisher>McMillin Publishing</Publisher>
          <bns:Reviewer bns:cod=“12”>
             <bns:name>
                <bns:First>Roger</bns:First>
                <bns:Last>Costello</bns:Last>
             </bns:name>
          </bns:Reviewer>
</Book>
<Book bns:id="R.B.">
          <Title>Illusions: The Adventures of a Reluctant Messiah</Title>
          <Author>Richard Bach</Author>
          <Date>1977</Date>
          <ISBN>0-440-34319-4</ISBN>
          <Publisher>Dell Publishing Co.</Publisher>
</Book>
</BookSeller>
```

Using a qualifier.
The qualifier indicates that we are using the *“www.books.org”* vocabulary

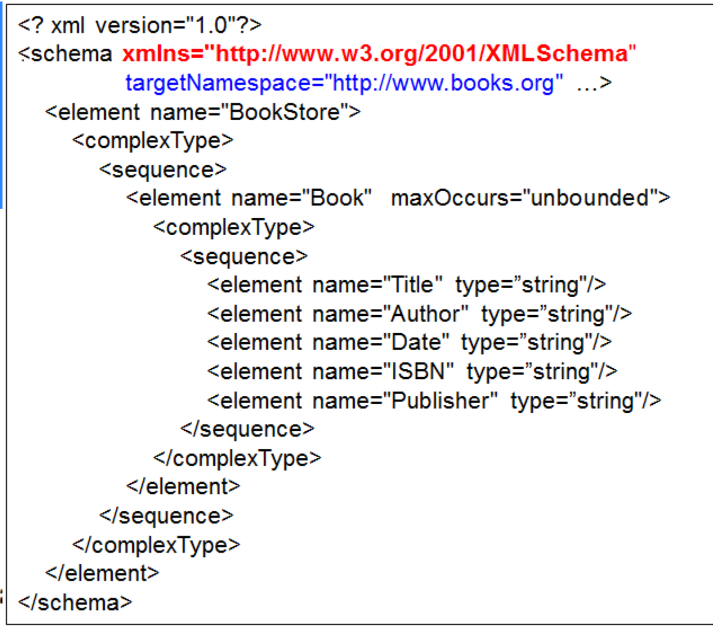
The subelements and attributes MUST be qualified, if they also come from the vocabulary of the element that contains them

informatika fakultatea facultad de informática

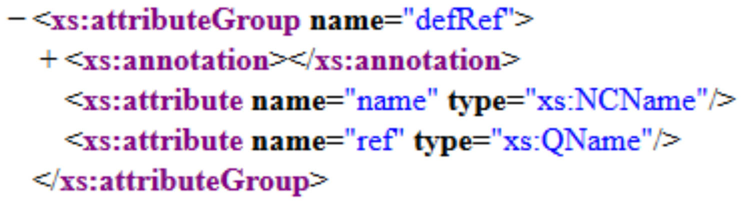
XML Schema



```
− <xs:complexType name="element" abstract="true">
  + <xs:annotation></xs:annotation>
  − <xs:complexContent>
    − <xs:extension base="xs:annotated">
      − <xs:sequence>
        − <xs:choice minOccurs="0">
            <xs:element name="simpleType" type="xs:localSimpleType"/>
            <xs:element name="complexType" type="xs:localComplexType"/>
          </xs:choice>
          <xs:element name="alternative" type="xs:altType" minOccurs="0" m
          <xs:group ref="xs:identityConstraint" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attributeGroup ref="xs:defRef"/>
        <xs:attribute name="type" type="xs:QName"/>
      − <xs:attribute name="substitutionGroup">
        − <xs:simpleType>
            <xs:list itemType="xs:QName"/>
          </xs:simpleType>
        </xs:attribute>
        <xs:attributeGroup ref="xs:occurs"/>
        <xs:attribute name="default" type="xs:string"/>
        <xs:attribute name="fixed" type="xs:string"/>
        <xs:attribute name="nillable" type="xs:boolean" use="optional"/>
        <xs:attribute name="abstract" type="xs:boolean" default="false" use="optional"/>
        <xs:attribute name="final" type="xs:derivationSet"/>
        <xs:attribute name="block" type="xs:blockSet"/>
        <xs:attribute name="form" type="xs:formChoice"/>
        <xs:attribute name="targetNamespace" type="xs:anyURI"/>
      </xs:extension>
    </xs:complexContent>
```



```
<? xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.books.org" ...>
  <element name="BookStore">
    <complexType>
      <sequence>
        <element name="Book"  maxOccurs="unbounded">
          <complexType>
            <sequence>
              <element name="Title" type="string"/>
              <element name="Author" type="string"/>
              <element name="Date" type="string"/>
              <element name="ISBN" type="string"/>
              <element name="Publisher" type="string"/>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>
```

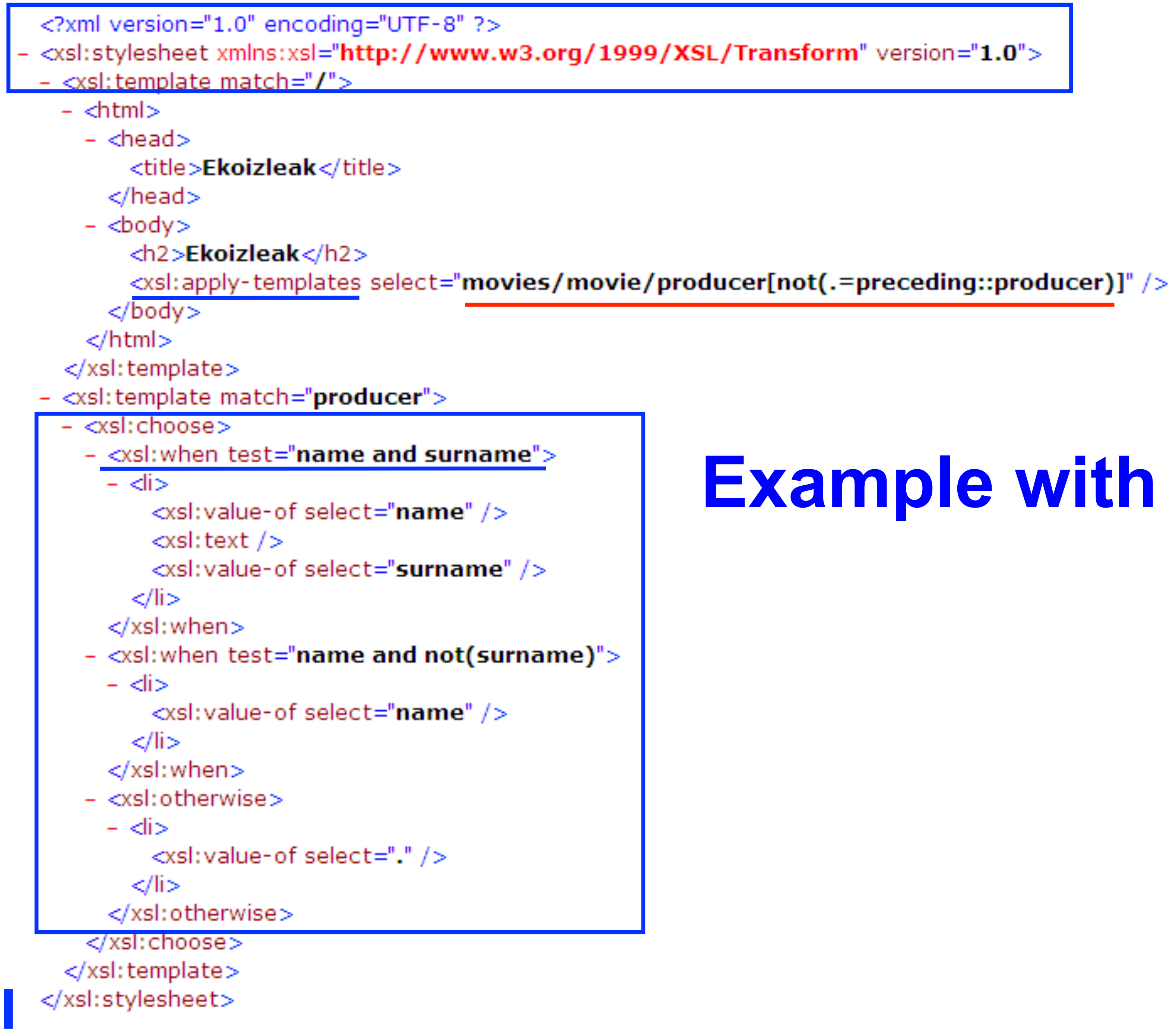


```
− <xs:attributeGroup name="defRef">
  + <xs:annotation></xs:annotation>
    <xs:attribute name="name" type="xs:NCName"/>
    <xs:attribute name="ref" type="xs:QName"/>
  </xs:attributeGroup>
```

■ ■ ■

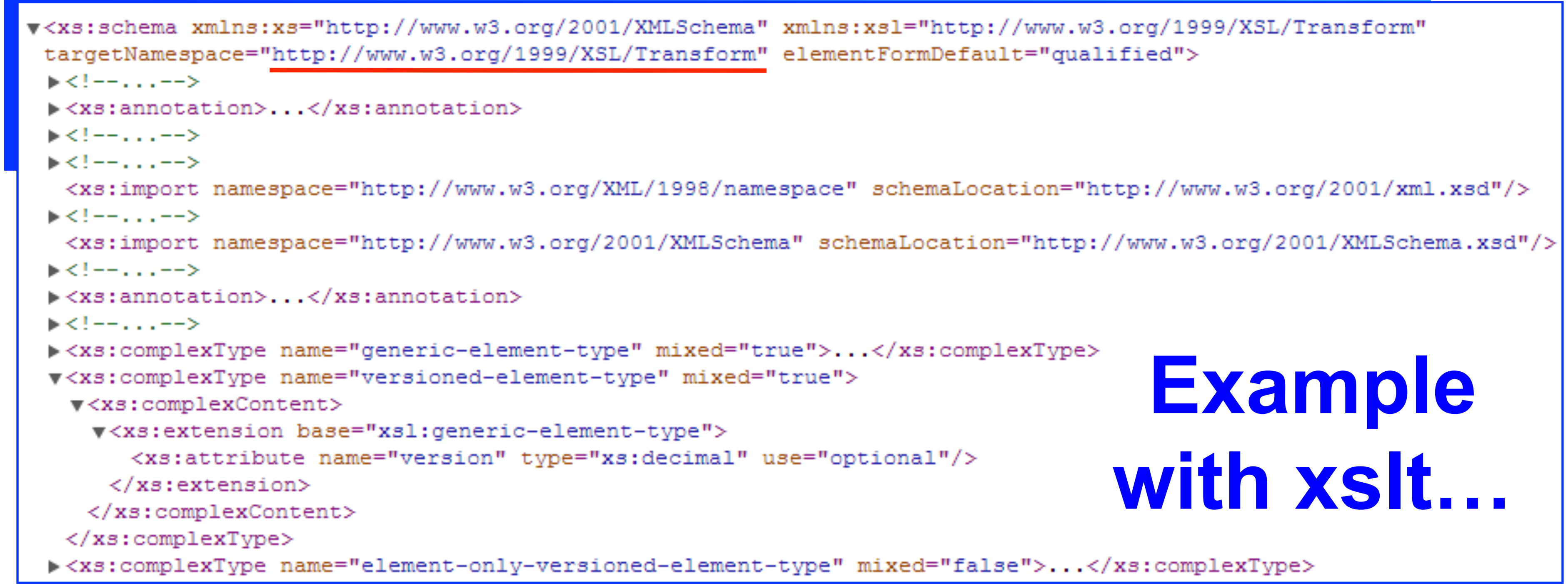
https://www.w3.org/2001/XMLSchema.xsd

Example with xslt…

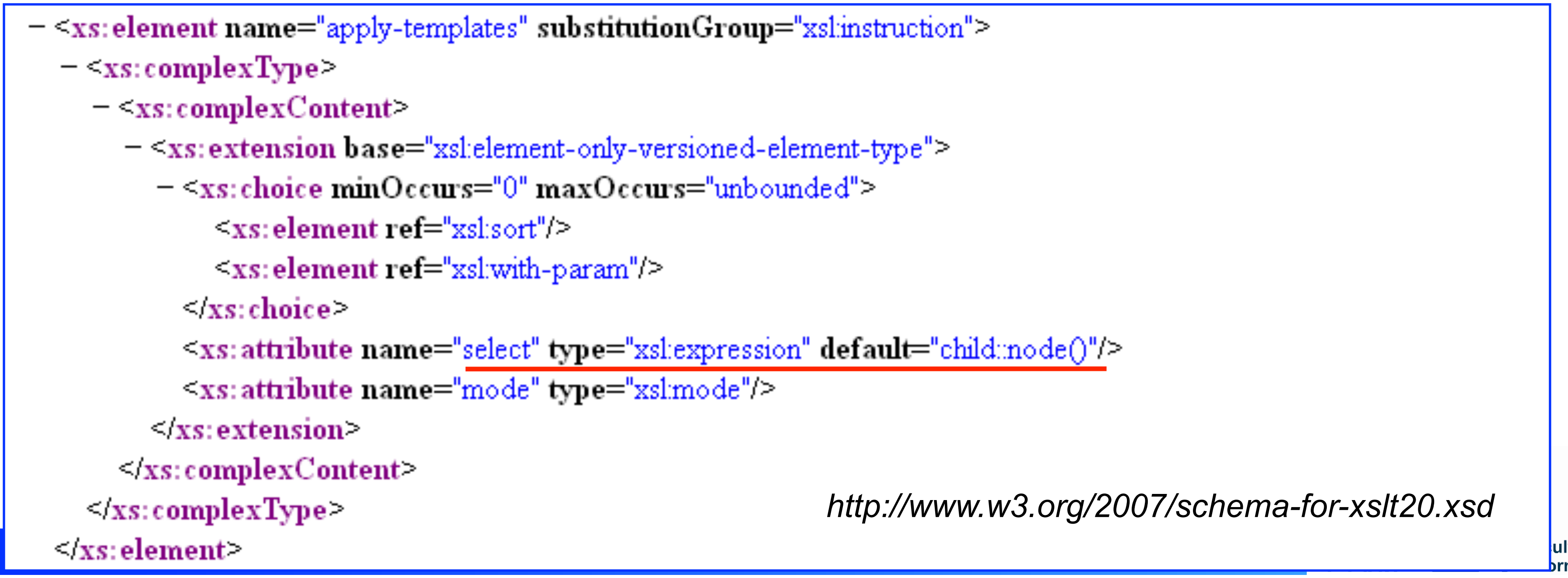


```
  <?xml version="1.0" encoding="UTF-8" ?>
- <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 - <xsl:template match="/">
  - <html>
   - <head>
       <title>Ekoizleak</title>
     </head>
   - <body>
       <h2>Ekoizleak</h2>
       <xsl:apply-templates select="movies/movie/producer[not(.=preceding::producer)]" />
     </body>
   </html>
  </xsl:template>
 - <xsl:template match="producer">
  - <xsl:choose>
   - <xsl:when test="name and surname">
    - <li>
        <xsl:value-of select="name" />
        <xsl:text />
        <xsl:value-of select="surname" />
      </li>
     </xsl:when>
   - <xsl:when test="name and not(surname)">
    - <li>
        <xsl:value-of select="name" />
      </li>
     </xsl:when>
   - <xsl:otherwise>
    - <li>
        <xsl:value-of select="." />
      </li>
     </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
 </xsl:stylesheet>
```

Example with xslt…



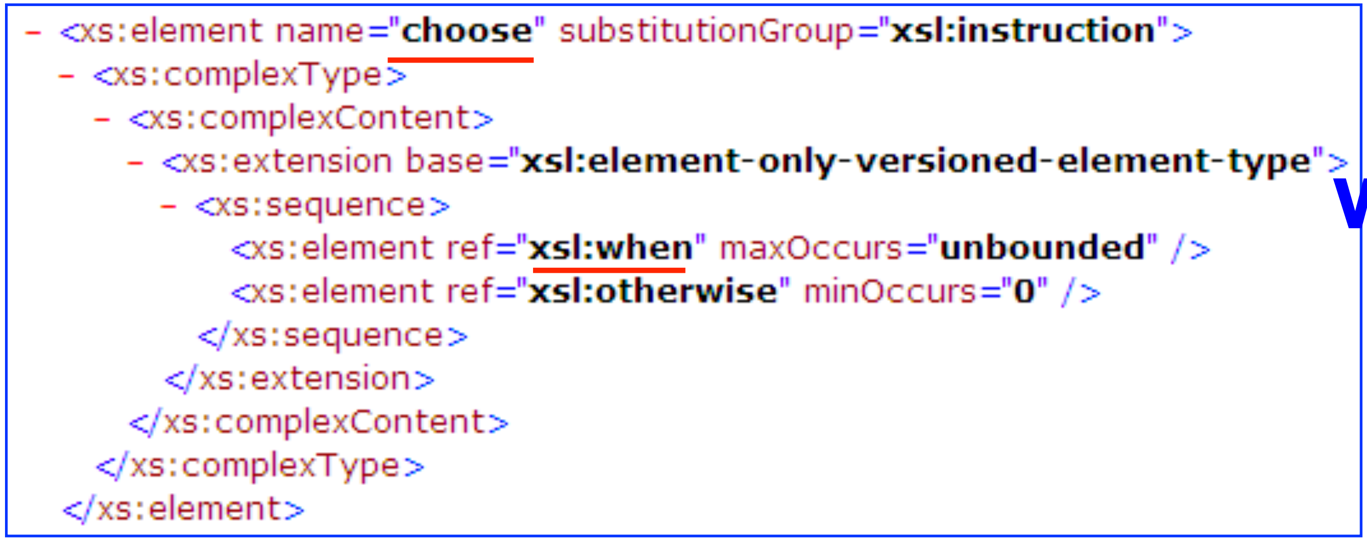
```
▼<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 targetNamespace="http://www.w3.org/1999/XSL/Transform" elementFormDefault="qualified">
  ▶<!--...-->
  ▶<xs:annotation>...</xs:annotation>
  ▶<!--...-->
  ▶<!--...-->
   <xs:import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation="http://www.w3.org/2001/xml.xsd"/>
  ▶<!--...-->
   <xs:import namespace="http://www.w3.org/2001/XMLSchema" schemaLocation="http://www.w3.org/2001/XMLSchema.xsd"/>
  ▶<!--...-->
  ▶<xs:annotation>...</xs:annotation>
  ▶<!--...-->
  ▶<xs:complexType name="generic-element-type" mixed="true">...</xs:complexType>
  ▼<xs:complexType name="versioned-element-type" mixed="true">
    ▼<xs:complexContent>
      ▼<xs:extension base="xsl:generic-element-type">
          <xs:attribute name="version" type="xs:decimal" use="optional"/>
        </xs:extension>
      </xs:complexContent>
   </xs:complexType>
  ▶<xs:complexType name="element-only-versioned-element-type" mixed="false">...</xs:complexType>
```



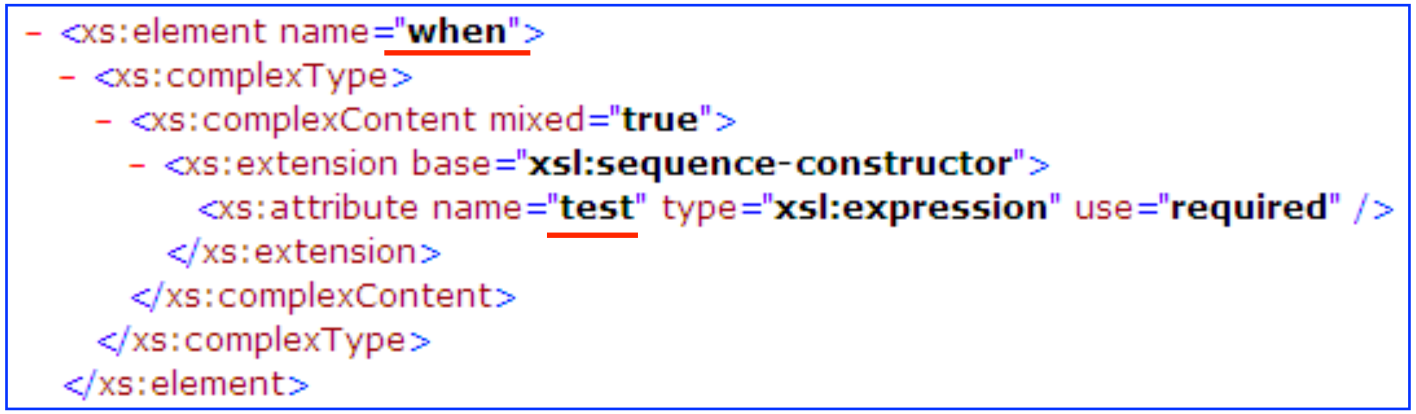
```
- <xs:element name="apply-templates" substitutionGroup="xsl:instruction">
  - <xs:complexType>
    - <xs:complexContent>
      - <xs:extension base="xsl:element-only-versioned-element-type">
        - <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element ref="xsl:sort"/>
            <xs:element ref="xsl:with-param"/>
          </xs:choice>
          <xs:attribute name="select" type="xsl:expression" default="child::node()"/>
          <xs:attribute name="mode" type="xsl:mode"/>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
```

http://www.w3.org/2007/schema-for-xslt20.xsd

Example with xslt…



```
- <xs:element name="choose" substitutionGroup="xsl:instruction">
  - <xs:complexType>
    - <xs:complexContent>
      - <xs:extension base="xsl:element-only-versioned-element-type">
        - <xs:sequence>
            <xs:element ref="xsl:when" maxOccurs="unbounded" />
            <xs:element ref="xsl:otherwise" minOccurs="0" />
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
```



```
- <xs:element name="when">
  - <xs:complexType>
    - <xs:complexContent mixed="true">
      - <xs:extension base="xsl:sequence-constructor">
          <xs:attribute name="test" type="xsl:expression" use="required" />
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
```

Contents

© Acknowledgments. Part of these slides have been prepared using R.L. Costello’s material

What is in an schema?

- Elements
- Atributes
- Simple types
- Complex types

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.books.org“ …>
 <xs:element name="BookStore">
   <xs:complexType>
     <xs:sequence>
       <xs:element name="Book"  maxOccurs="unbounded">
         <xs:complexType>
           <xs:sequence>
             <xs:element name="Title" type=”string"/>
             <xs:element name="Author" type=”string"/>
             <xs:element name="Date" type=”string"/>
             <xs:element name="ISBN" type=”string"/>
             <xs:element name="Publisher" type=”string"/>
           </xs:sequence>
           <xs:attribute name="href" type=”xs:anyURI" use="required"/>
         </xs:complexType>
       </xs:element>
     </xs:sequence>
   </xs:complexType>
 </xs:element>
</xs:schema>
```

Attributes: declaration

1

```
<xs:attribute name=“name" type=“simple-type" use=“how-it-is-used" default/fixed="value"/>
```

xs:string
xs:integer
xs:boolean
...

required
optional
prohibited

The "use" attribute must be “optional”, if “default” or “fixed” are used

2

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

Attributes: Examples

```
<xs:attribute name="language" type="xs:string" default="EN"/>

<xs:attribute name="language" type="xs:string" fixed="EN"/>

<xs:attribute name="language" type="xs:string" use="required"/>
```

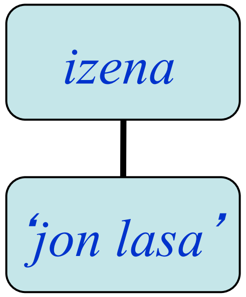
Element: declaration

- Simple element
  - Only text
  - No attributes, neither subelements
- Complex element
  - Four kinds
    - empty elements
    - elements that contain only other elements
    - elements that contain only text
    - elements that contain both other elements and text. Each of these elements
  - All of them may content attributes

Definition:
- Inline
- By reference
- With type

Simple element

```
<izena>jon lasa</izena>
```



```
<xs:element name="izena" type="mota” />
```

Name of the element

data type of the element. XML Schema has a lot of built-in data types (xs:string, xs:decimal, xs:integer, xs:boolean, xs:date, xs:time)

Complex element

```
<product pid="1345"/>
```

```
<employee>
    <firstname>John</firstname>
    <lastname>Smith</lastname>
</employee>
```

```
<food type="dessert">Ice cream</food>
```

```
<description>
    It happened on <date lang=“euskera">12-10-30</date>
   ....
</description>
```

Complex element: Empty

```
<img href="http://www.xfront.com/InSubway.gif"/>
```

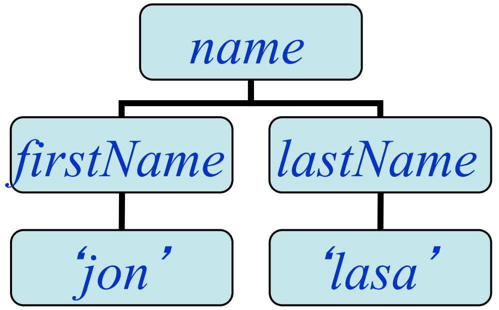
Instance example

```
<xs:element name="img">
    <xs:complexType>
        <xs:attribute name="href" type="xs:anyURI" use="required"/>
    </xs:complexType>
</xs:element>
```

An empty element does not have content but it can have attributes

Complex element: with subelements

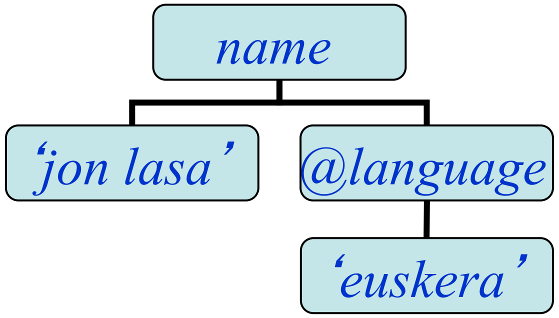
```
<name>
        <firstName>jon</firstName>
        <lastName>lasa</lastName>
</name>
```



```
<xs:element name="name">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="firstName" type="xs:string"/>
            <xs:element name="lastName" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Complex element: with text

```
<name language=“euskera”>
            jon lasa
</name>
```



```
<xs:element name=“name">
   <xs:complexType>
         <xs:simpleContent>
              <xs:extension base="xs:string">
                   <xs:attribute name=“language" type="xs:string" />
              </xs:extension>
         </xs:simpleContent>
   </xs:complexType>
</xs:element>
```

Complex element: mixed content

Content of an element can be:

- a basic value (e.g. a string, an integer)
- other sub-elements
- a mixture of both

```
<book isbn="0836217462">Nafarroa Behereko Garaziko eskualdekoa
zen <author>Bernat Etxepare</author>, eta 1545ean Bordelen
plazaratu zuen bere <title>Linguae vasconum primitiae</title> liburua.
Euskarari "kanpora, plazara dantzara" irteteko agintzen dio, eta....
</book>
```

Complex element: mixed content (2)

```
<book>Nafarroa Behereko Garaziko eskualdekoa zen
<author>Bernat Etxepare</author>, eta 1545ean Bordelen plazaratu
zuen bere <title>Linguae vasconum primitiae</title> liburua. Euskarari
"kanpora, plazara dantzara" irteteko agintzen dio, eta....
</book>
```

Schema definition

```
<xs:element name="book"
    <xs:complexType mixed="true">
        <xs:sequence>
                <xs:element name="author" type="xs:string"/>
                <xs:element name="title" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Element definition: Inline definition (a.k.a russian-doll approach)

- Element and type definition is done inside an element
  - Title
  - Book's type
- Drawback: the definition of elements cannot be reused

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.books.org“ …>
  <element name="BookStore">
    <complexType>
      <sequence>
        <element name="Book"  maxOccurs="unbounded">
          <complexType>
            <sequence>
              <element name="Title" type="xs:string"/>
              <element name="Author" type="xs:string"/>
              <element name="Date" type="xs:string"/>
              <element name="ISBN" type="xs:string"/>
              <element name="Publisher" type="xs:string"/>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>
```

Element definition: by reference

- The element is defined separately
- It is used by reference
- It can be reused

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                  targetNamespace="http://www.books.org"
                  xmlns="http://www.books.org"
                  elementFormDefault="qualified">
  <xs:element name="BookStore">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Book” type=“BookT”/>

  <xs:element name="Title" type="xs:string"/>
  <xs:element name="Author" type="xs:string"/>
  <xs:complexType name=“BookT”>
      <xs:sequence>
        <xs:element ref="Title" minOccurs="1" maxOccurs="1"/>
        <xs:element ref="Author" minOccurs="1" maxOccurs="1"/>
       </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Element definition: with type

- A complex type is defined separately
- The element refers to the name of the complex type
- It can be reused
- A complex type can base on another existing complex type and add some elements

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.books.org"
           xmlns="http://www.books.org"
           elementFormDefault="qualified">
  <xs:element name="BookStore">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Book" type="BookT"/>

  <xs:element name="Title" type="xs:string"/>
  <xs:element name="Author" type="xs:string"/>
  <xs:complexType name="BookT">
      <xs:sequence>
        <xs:element ref="Title" minOccurs="1" maxOccurs="1"/>
        <xs:element ref="Author" minOccurs="1" maxOccurs="1"/>
       </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Data Types

- Named types
  - Those that have a name, and they are used by reference

```
<xs:complexType name="BookT">
      <xs:sequence>
      …
      </xs:sequence>
</xs:complexType>

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

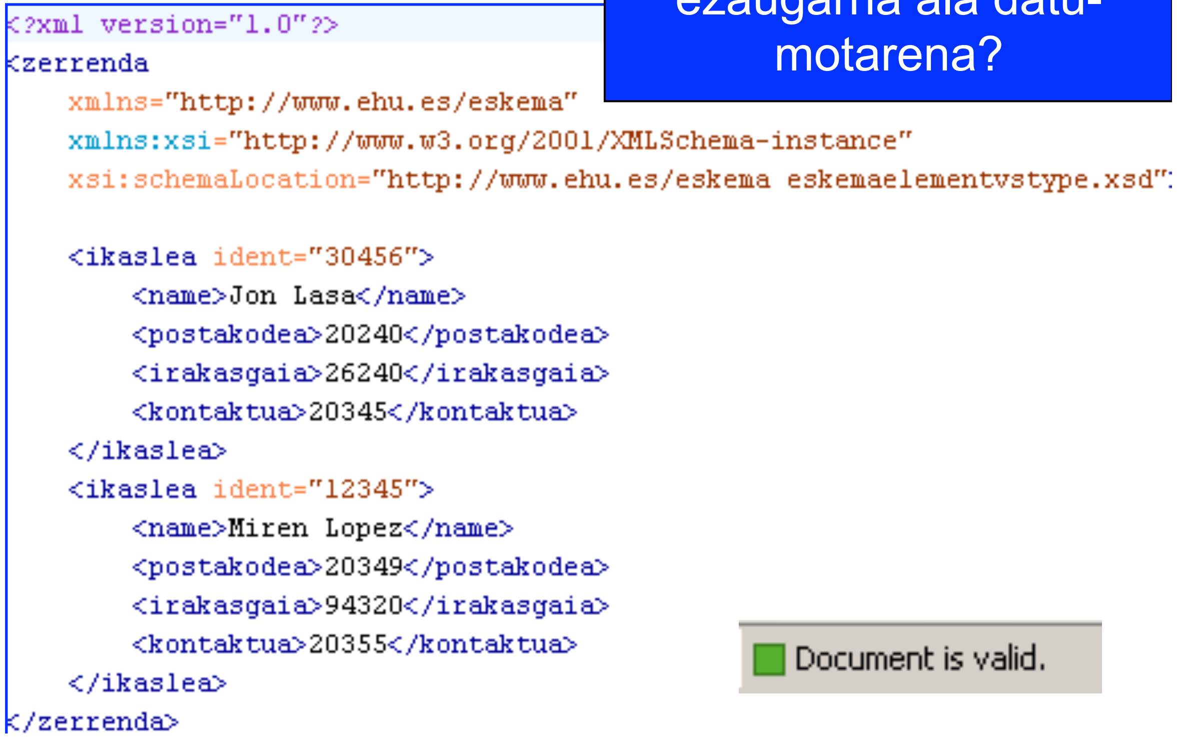
- Anonymous types
  - Those that have no name, and they are used in line

```
<xs:element name="Book">
      <xs:complexType>
          <xs:sequence>

          </xs:sequence>
      </xs:complexType>
</xs:element>
```

Data type vs. Element

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"    targetNamespace="http://www.ehu.es/eskema"
      xmlns="http://www.ehu.es/eskema"   elementFormDefault="qualified">
   <xs:element name="zerrenda">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="ikaslea" minOccurs="1" maxOccurs="unbounded">
               <xs:complexType mixed="true">
                  <xs:sequence>
                     <xs:element name="name" type="xs:string"/>
                     <xs:element name="postakodea" type="bostdigituT“ default=“20000”/>
                     <xs:element name="irakasgaia" type="bostdigituT"/>
                     <xs:element name="kontaktua" type="gipuzkoaT"/>
                  </xs:sequence>
                  <xs:attribute name="ident" type="bostdigituT" use="required"/>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:simpleType name="bostdigituT">
      <xs:restriction base="xs:string">
           <xs:pattern value="\d{5}"/>
      </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="gipuzkoaT">
     <xs:restriction base="bostdigituT">
         <xs:pattern value="20\d{3}"/>
     </xs:restriction>
   </xs:simpleType>
</xs:schema>
```

Defektuzko balioa, ‘postakodea’ -ren ezaugarria ala datu-motarena?

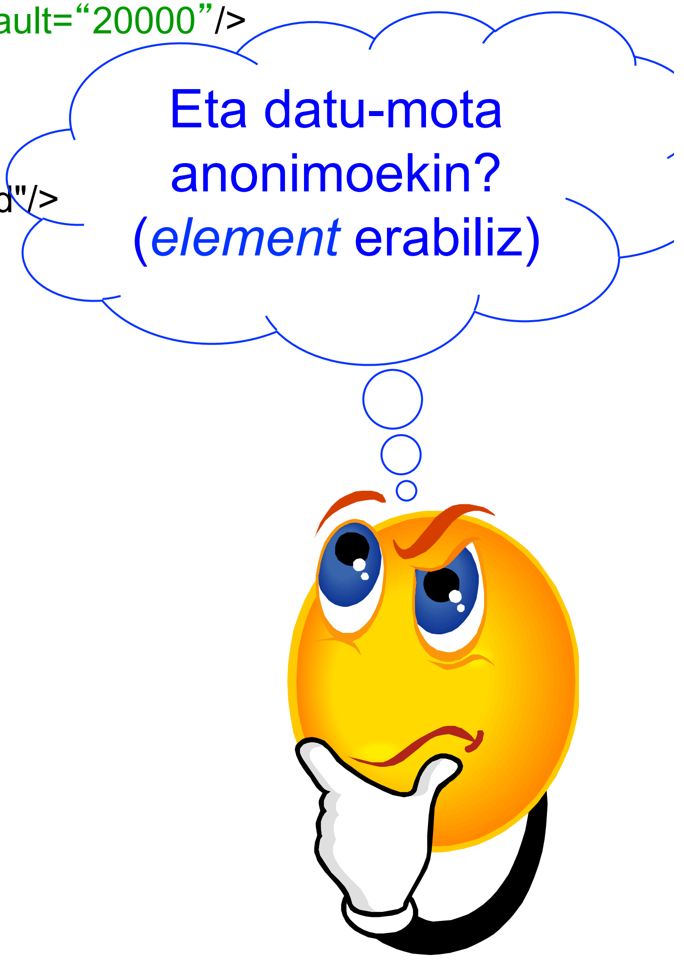


```
<?xml version="1.0"?>
<zerrenda
    xmlns="http://www.ehu.es/eskema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.ehu.es/eskema eskemaelementvstype.xsd"

    <ikaslea ident="30456">
        <name>Jon Lasa</name>
        <postakodea>20240</postakodea>
        <irakasgaia>26240</irakasgaia>
        <kontaktua>20345</kontaktua>
    </ikaslea>
    <ikaslea ident="12345">
        <name>Miren Lopez</name>
        <postakodea>20349</postakodea>
        <irakasgaia>94320</irakasgaia>
        <kontaktua>20355</kontaktua>
    </ikaslea>
</zerrenda>
```

🟩 Document is valid.

Data type vs. Element

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"    targetNamespace="http://www.ehu.es/eskema"
        xmlns="http://www.ehu.es/eskema"   elementFormDefault="qualified">
    <xs:element name="zerrenda">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="ikaslea" minOccurs="1" maxOccurs="unbounded">
                    <xs:complexType mixed="true">
                        <xs:sequence>
                            <xs:element name="name" type="xs:string"/>
                            <xs:element name="postakodea" type="bostdigituT“ default=“20000”/>
                            <xs:element name="irakasgaia" type="bostdigituT"/>
                            <xs:element name="kontaktua" type="gipuzkoaT"/>
                        </xs:sequence>
                        <xs:attribute name="ident" type="bostdigituT" use="required"/>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:simpleType name="bostdigituT">
        <xs:restriction base="xs:string">
              <xs:pattern value="\d{5}"/>
        </xs:restriction>
    </xs:simpleType>
  <xs:simpleType name="gipuzkoaT">
       <xs:restriction base="bostdigituT">
            <xs:pattern value="20\d{3}"/>
       </xs:restriction>
    </xs:simpleType>
</xs:schema>
```



Data type vs. Element

```
?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.ehu.es/eskema"
        xmlns="http://www.ehu.es/eskema"
        elementFormDefault="qualified">
    <xs:element name="zerrenda">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="ikaslea" minOccurs="1" maxOccurs="unbounded">
                    <xs:complexType mixed="true">
                        <xs:sequence>
                            <xs:element name="name" type="xs:string"/>
                            <xs:element ref="postakodea"/>
                            <xs:element name="irakasgaia"  …../>
                            <xs:element name="kontaktua" />
                        </xs:sequence>
                      <xs:attribute name="ident" …. />
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="postakodea“ default=“20000”>
       <xs:simpleType>
           <xs:restriction base="xs:string">  <xs:pattern value="\d{5}"/>  </xs:restriction>
       </xs:simpleType>
   </xs:element>
   <xs:simpleType name="gipuzkoaT">
       <xs:restriction base=“postakodea">  <xs:pattern value="20\d{3}"/>   </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

‘irakasgaia’, ‘kontaktua’, ‘ident’ nola definitzen ditugu?

Defektuzko balioa, ‘postakodea’ erreferentziatzen duten elementu guztietarako !!!

Errorea!!

Types

- Simple types
  - Basic types (pre-defined by XML Schema)
  - Simple types, derived from basic types
  - Simple types, derived from derived simple types
  - Simple types, obtained as lists/unions
  - ID and IDREF
- Complex types
  - Complex types with attributes or subelements
  - Complex derived types with simple content
  - Complex derived types with complex content
  - Complex types with mixed content

Simple types

- For elements
  - Only have content
  - No structure
  - No attributes
- For attributes

Example: only *ISBN*, *First* and *Last* are simple type elements

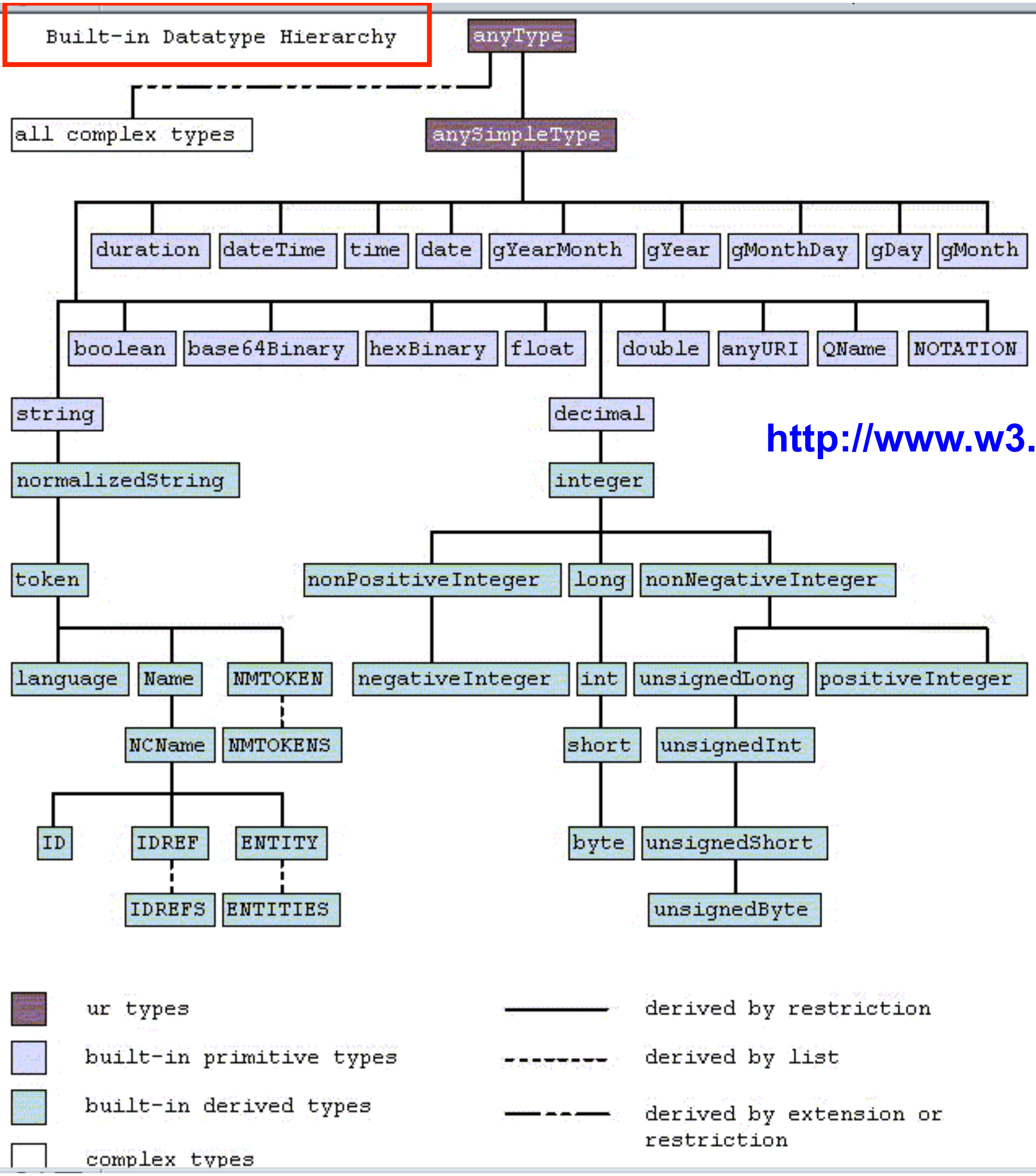
```
<ISBN>1-56592-235-2</ISBN>
<Publisher country = “Spain” >McMillan Publishing</Publisher>
<name>
      <First>Roger</First>
      <Last>Costello</Last>
 </name>
```

Simple types: some basic types

string	**"Hello World"**
boolean	{**true**, **false**, **1**, **0**}
decimal	**7.08**
float	**12.56E3**, **12**, **12560**, **0**, **-0**, **INF**, **-INF**, **NAN**
double	**12.56E3**, **12**, **12560**, **0**, **-0**, **INF**, **-INF**, **NAN**
duration	**P1Y2M3DT10H30M12.3S**
dateTime	format: *CCYY-MM-DDThh:mm:ss*
time	format: *hh:mm:ss.sss*
date	format: *CCYY-MM-DD*
gYearMonth	format: *CCYY-MM*
gYear	format: *CCYY*
gMonthDy	format: --*MM-DD*

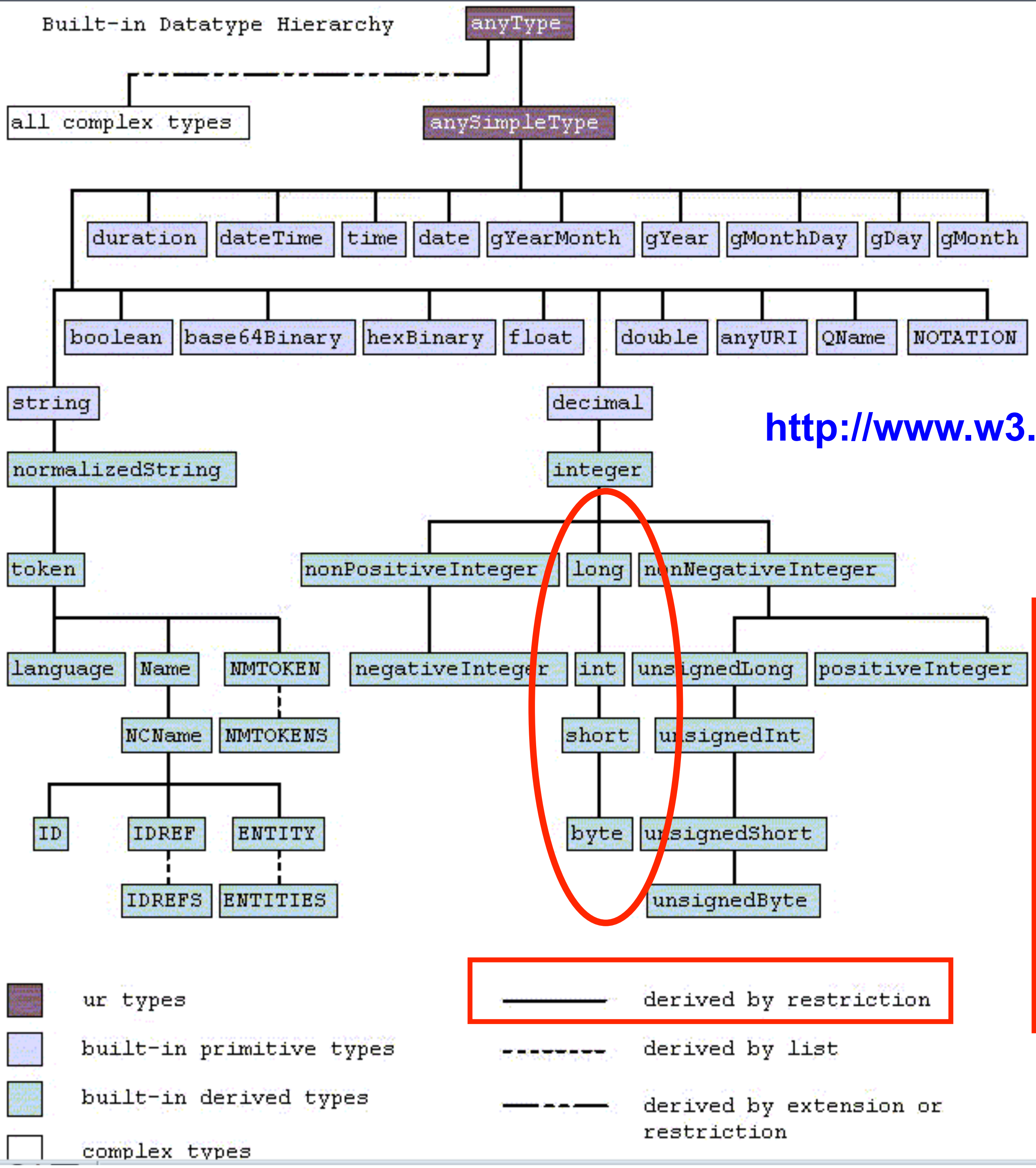
Note: 'T' is the date/time separator
INF = infinity
NAN = not-a-number

W3C Recommendation

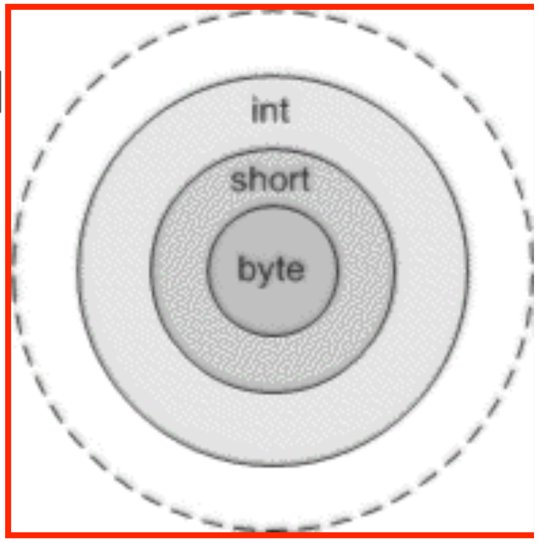


http://www.w3.org/2001/XMLSchema

W3C Recommendation



http://www.w3.org/2001/XMLSchema



SimpleType. Syntax

```
<simpleType
    final = (#all | List of (list | union | restriction))
    id = ID
    name = NCName
    {any attributes with non-schema namespace . . .}>
    Content:
        (annotation?, (restriction | list | union))
</simpleType>
```

Simple types: lists

- Defined using the “list” (meta) element

```
<xs:simpleType name="listOfMyIntT">
    <xs:list itemType="myInteger"/>
</xs:simpleType>
```

```
<element name=“myList” type=“listOfMyIntT” />

<myList>20003 15037 95977 95945</myList>
```

Simple Types: Derived <restriction>

- Each type has a set of valid values
- A type can be derived from another type (the "base" type) by restricting its value set
- This restriction is expressed in terms of "facets" of the basic type
  - Example "facets" of the "string" type
    - length
    - minLength
    - maxLength
    - pattern
    - enumeration
    - whitespace ({preserve, replace, collapse})

informatika fakultatea facultad de informática

Simple types: Derived. Syntax

```
<xs:simpleType name= "name">
   <xs:restriction base= "xs:source">
      <xs:facet value= "value"/>
      <xs:facet value= "value"/>
      …
   </xs:restriction>
</xs:simpleType>
```

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

...

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

...

Types derived from built-in types “*String*” type facets. Example

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

An element of type “*shapeT*” can only hold a string out of {circle, triangle, square}

informatika fakultatea facultad de informática

Types derived from built-in types “*String*” type facets. Example (2)

```
<xs:simpleType name="TelephoneNumberT">
   <xs:restriction base="xs:string">
        <xs:length value="10"/>
        <xs:pattern value="\d{3}-\d{6}"/>
   </xs:restriction>
</xs:simpleType>
```

- The new *‘TelephoneNumberT’* type is created
- Elements of this type contain “*strings*”
- But the length of the *“string”* is restricted to 10 characters, and
- The *“string”* must follow the ddd-dddddd pattern, where 'd' corresponds to a 'digit'
  (Note: In this example the regular expression makes the length restriction redundant)

informatika fakultatea facultad de informática

Types derived from built-in types “*String*” type facets. Example (3)

```
<xs:simpleType name = "passwordType">
        <xs:restriction base="xs:string">
                <xs:pattern value="[a-zA-Z0-9]{8}"/>
        </xs:restriction>
</xs:simpleType> >
```

The regular expression restricts passwords to be eight-length alpha-numeric characters

Types derived from built-in types
Some String pattern examples

Regular expression	Example
Chapter \d	Chapter 1
a*b	b, ab, aab, aaab, …
[xyz]b	xb, yb, zb
a?b	b, ab
a+b	ab, aab, aaab, …
[a-c]x	ax, bx, cx
[-ac]x	-x, ax, cx
[ac-]x	ax, cx, -x
[^0-9]x	*any non-digit char followed by x*
\Dx	*any non-digit char followed by x*
Chapter\s\d	*Chapter* followed by a blank followed by a digit
(ho){2} there	hoho there
(ho\s){2} there	ho ho there
.abc	*any (one) char followed by abc*
(a|b)+x	ax, bx, aax, bbx, abx, bax,...

Types derived from built-in types. "*String*" type facets: Whitespace

- It controls how white space in the element will be processed
- There are three possible values
  - **"preserve"** causes the processor to keep all whitespace as-is
  - **"replace"** causes the processor to replace all whitespace characters (tabs, carriage returns, line feeds, spaces) with space characters
  - **"collapse"** causes the processor to replace all strings of whitespace characters (tabs, carriage returns, line feeds, spaces) with a single space character

```
<xs:simpleType name="addressType>
        <xs:restriction base="xs:string">
                <xs:whitespace value="replace"/>
        </xs:restriction>
</xs:simpleType>
```

Types derived from built-in types “*Integer*” type facets. Example

```
<xs:element name=“prezioa” type=“prezioaT” />

<xs:simpleType name=“prezioaT”>
    <xs:restriction base="xs:integer">
        <xs:minInclusive value=“1000”/>
        <xs:maxInclusive value=“10000”/>
    </xs:restriction>
</xs:simpleType>
```

We want to restrict possible values of the element

```
<prezioa>5440</prezioa>
```

✓

```
<prezioa>540</prezioa>
```

✗

Derived types. “*Integer*” type facets

Facet	Description
enumeration	Defines a list of acceptable values
fractionDigits	The maximum number of decimal places allowed. >=0
length	The exact number of characters or list items allowed. >=0
maxExclusive	The upper bounds for numeric values (the value must be less than the value specified)
maxInclusive	The upper bounds for numeric values (the value must be less than or equal to the value specified)
maxLength	The maximum number of characters or list items allowed. >=0
minExclusive	The lower bounds for numeric values (the value must be greater than the value specified)
minInclusive	The lower bounds for numeric values (the value must be greater than or equal to the value specified)
minLength	The minimum number of characters or list items allowed >=0
pattern	The sequence of acceptable characters based on a regular expression
totalDigits	The exact number of digits allowed. >0
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

Types derived from other simple types

- A derived type can be used as the “base” type
- The new type must be more restrictive than the “base” type

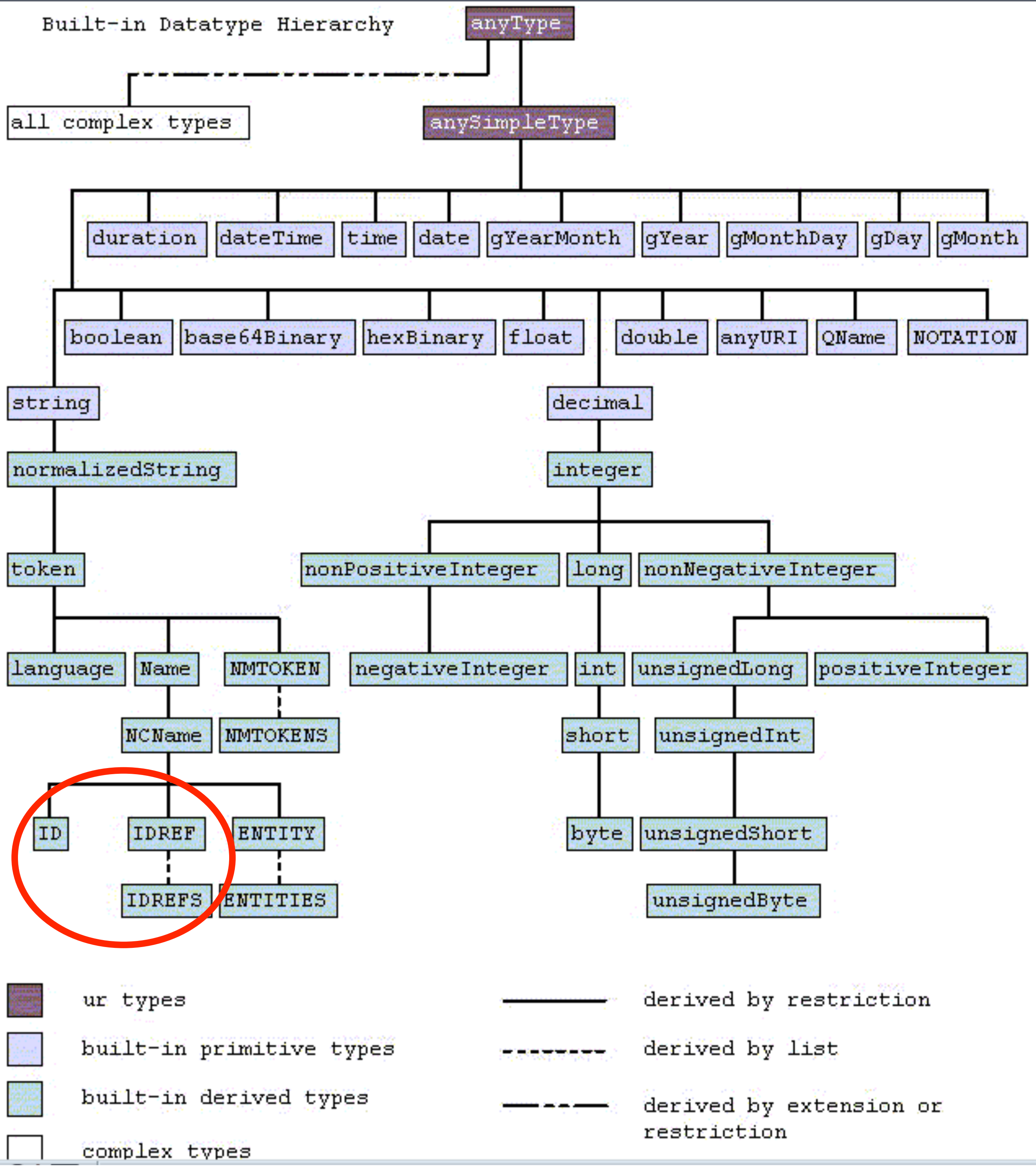
```
<xs:simpleType name= “latitudeaT">
      <xs:restriction base="xs:integer">
            <xs:minInclusive value="-90"/>
            <xs:maxInclusive value="90"/>
      </xs:restriction>
</xs:simpleType>
```

```
<xs:simpleType name= “latitudeaEHT">
      <xs:restriction base="latitudeaT">
            <xs:minInclusive value=”42"/>
            <xs:maxInclusive value="44"/>
      </xs:restriction>
</xs:simpleType>
```

Simple derived types
Setting the value of a “facet”

```
<xs:simpleType name= “ikasleAdinaT">
    <xs:restriction base="xs:nonNegativeInteger">
        <xs:minInclusive value="18" fixed="true"/>
        <xs:maxInclusive value=“90"/>
   </xs:restriction>
</xs:simpleType>
```

Those types derived from *ikasleAdinaT* cannot change the lower limit


Built-in Datatype Hierarchy
anyType
all complex types
anySimpleType
duration
dateTime
time
date
gYearMonth
gYear
gMonthDay
gDay
gMonth
boolean
base64Binary
hexBinary
float
double
anyURI
QName
NOTATION
string
decimal
normalizedString
integer
token
nonPositiveInteger
long
nonNegativeInteger
language
Name
NMTOKEN
negativeInteger
int
unsignedLong
positiveInteger
NCName
NMTOKENS
short
unsignedInt
ID
IDREF
ENTITY
byte
unsignedShort
IDREFS
ENTITIES
unsignedByte
ur types
built-in primitive types
built-in derived types
complex types
derived by restriction
derived by list
derived by extension or restriction

Types ID and IDREF

- ID type restricts the value to be unique <u>within the whole document</u>

- IDREF restricts the value to coincide with another value that is ID typed
  - IDREFS, the same but with a list of values

ID and IDREF: Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xs:element name="orders">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="order" type="orderDetails" />
        <xs:element name="orderlist" type="orderLists" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="orderDetails">
    <xs:sequence>
      <xs:element name="customerName" type="xs:string"/>
      <xs:element name="customerAddress" type="xs:string"/>
      <xs:element name="customerContact" type="xs:string"/>
      <xs:element name="orderIDREF" type="xs:IDREF"/>
      <xs:element name="orderIDREFS" type="xs:IDREFS"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="orderLists">
    <xs:sequence>
      <xs:element name="orderID" type="xs:ID" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<orders>
  <order>
    <customerName>Test</customerName>
    <customerAddress>Test Address</customerAc
    <customerContact>12345678</customerConta
    <orderIDREF>k1</orderIDREF>
    <orderIDREFS>k1 k2</orderIDREFS>
  </order>
  <orderlist>
    <orderID>k1</orderID>
    <orderID>k2</orderID>
  </orderlist>
</orders>
```

ID/IDREFS vs. Database keys

- ID unique within the entire document (like oids),
  - while a key needs only to uniquely identify a tuple within a relation
- IDREF untyped: one has no control over what it points to
  - You point to something, but you don't know what it is!

```
<student  id="01"  name="John"  taking="CS2"/>
<student  id="02"  name="Peter"   taking="01"/>
<course   id="CS2"/>
```

- IDs are based on a single element
  - While keys can be based on more than one attribute (e.g. enroll (<u>sid: string, cid: string</u>, grade: string))
- An element can have at most one ID (primary)
  - While a relation may have multiple keys

ID/IDREFS vs. Database keys Example

```
<xs:element name="school">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="student" maxOccurs="unbounded">
                    <xs:complexType>
                            <xs:attribute name="id" type="xs:ID" />
                            <xs:attribute name="name" type="xs:string" />
                            <xs:attribute name="taking" type="xs:IDREF" />
                    </xs:complexType>
                </xs:element>
                <xs:element name="course" maxOccurs="unbounded">
                    <xs:complexType>
                            <xs:attribute name="id" type="xs:ID" />
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

Types

- Simple types
  - Basic types (pre-defined by XML Schema)
  - Simple types, derived from basic types
  - Simple types, derived from derived simple types
  - Simple types, obtained as lists/unions
  - ID and IDREF
- Complex types
  - Complex types with attributes or subelements
  - Complex derived types with extension
  - Complex derived types with restriction
  - Type Substitution
  - Control mechanisms for type derivation and substitution

ComplexType. Syntax

```
<complexType
    id=ID
    name=NCName
    abstract=true | false
    mixed=true | false
    block=(#all | list of (extension | restriction))
    final=(#all | list of (extension | restriction))
    any attributes
>
    (annotation?,
        (simpleContent | complexContent |
            ((group | all | choice| sequence)?,
                ((attribute | attributeGroup)*, anyAttribute?))))

</complexType>
```

Derived types

Structured types

Structured complex types

- Sub-elements are structured through restrictions …
  - *<all>*: all subelements must be present
  - *<sequence>*: all subelements must be present in a given order
  - *<choice>*: there are several choices of subelements
- … and occurrence indicators (how often an element can occur)
  - *minOccurs* and *maxOccurs*

Structured complex types: minOccurs / maxOccurs indicators

- **maxOccurs** indicator specifies the maximum number of times an element can occur
- **minOccurs** indicator specifies the minimum number of times an element can occur
- Default values: 1

	minOccurs	**maxOccurs**
sequence	0/1/…	1/.../unbounded
all	0/1	1
choice	0/1/…	1/.../unbounded

Structured types: <sequence>

Sequence: the order is meaningful

```
<xs:complexType name=“ikasleaT">
      <xs:sequence>
            <xs:element name=“izena" type="xs:string"/>
            <xs:element name=“abizena" type="xs:string"/>
            <xs:element name=“adina" type=“adinaT"/>
            <xs:element name=“ikasmaila" type=“mailaT"/>
      </xs:sequence>
</xs:complexType>
```

Cardinality restrictions can be set

```
<xs:complexType name=“ikasleaT">
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name=“izena" type="xs:string"/>
            <xs:element name=“abizena" type="xs:string"/>
            <xs:element name=“adina" type=“adinaT"/>
            <xs:element name=“ikasmaila" type=“mailaT"/>
      </xs:sequence>
</xs:complexType>
```

Structured types: <choice>

Choice: alternatives

```
<xs:complexType name="komunikabideaT"> >
      <xs:choice>
          <xs:element name="telefonoa" type="telefonoT"/>
          <xs:element name="postaElektronikoa" type="epostaT"/>
          <xs:element name="postaHelbidea" type="helbideT"/>
      </xs:choice>
</xs:complexType>
```

Structured types: <all>

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.books.org"
           xmlns="http://www.books.org"
           elementFormDefault="qualified">
  <xs:element name="BookStore">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Book"  maxOccurs="unbounded">
          <xs:complexType>
            <xs:all>
              <xs:element name="Title" type="xs:string"/>
              <xs:element name="Author" type="xs:string"/>
              <xs:element name="Date" type="xs:string"/>
              <xs:element name="ISBN" type="xs:string"/>
              <xs:element name="Publisher" type="xs:string"
            </xs:all>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

A book must have the 5 elements, in any order

- Elements inside <all>: maxOccurs = "1“, minOccurs = "0" or "1“
- <all> cannot be nested to <sequence>, <choice>, or another <all>
- Content of <all> must be elements. <sequence> and <choice> are NOT allowed

Structured types: <*sequence*> & <choice>

Sequence, choice: they can be nested

```
<xs:complexType name="lifeT"> >
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:sequence minOccurs="0" maxOccurs="unbounded">
                <xs:element name="work" type="xs:string"/>
                <xs:element name="eat" type="xs:string"/>
        </xs: sequence>
        <xs:choice>
                <xs:element name=“read" type="xs:string"/>
                <xs:element name="play" type="xs:string"/>
        </xs:choice>
        <xs:element name="sleep" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name=“category“ type=“xs:string” use="required"/>
</xs:complexType>
```

Attributes are defined *after* sub-elements

Complex derived types

- A new type can be derived …
  - extending an existing type: **<extension>**
    - simpleType or complexType
    - ➡ The result is a complexType
  - restricting an existing type: **<restriction>**
    - simpleType, simpleContent, or complexContent
    - ➡ The result is a simpleType or a complexType (with simpleContent or complexContent)
- Substitution mechanisms between parent-derived types

Complex derived types: extension

```
<xs:complexType name="…">
    <xs:complexContent>
        <xs:extension base="X">
            …
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

```
<xs:complexType name="…">
    <xs:simpleContent>
        <xs:extension base="Y">
            …
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
```

X must be a *complexType with complexContent*

Y must be a *simpleType* or *complexType with simpleContent*

complexContent: defines extensions or restrictions on a complex type that contains mixed content or elements only

simpleContent: contains extensions or restrictions on a text-only complex type or on a simple type as content and contains no elements

Types derived from simple types: <extension>

```
<xs:element name="prezioa" type="prezioaT"/>
<xs:complexType name="prezioaT">
    <xs:simpleContent>
          <xs:extension base="xs:integer">
             <xs:attribute name="currency" type="xs:string" use="required"/>
          </xs:extension>
    </xs:simpleContent>
</xs:complexType>
```

We want to extend "integer" with a "currency" attribute

Example of element with ***prezioaT* type**

```
<prezioa currency="dolar">5440</prezioa>
```

informatika fakultatea facultad de informática

Types derived from complex types: <extension>

```
<xs:complexType name="PublicationT">
    <xs:sequence>
      <xs:element name=“title" type="xs:string" maxOccurs="unbounded"/>
      <xs:element name=“author" type="xs:string" maxOccurs="unbounded"/>
      <xs:element name=“date" type="xs:gYear"/>
    </xs:sequence>
</xs:complexType >

<xs:complexType name="BookPublicationT">
   <xs:complexContent>
      <xs:extension base="PublicationT">
         <xs:sequence>
              <xs:element name="ISBN" type="xs:string"/>
              <xs:element name=“publisher" type="xs:string"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType >
```

BookPublicationT has the elements of *PublicationT* + its own

The latter are always added at the end

Types derived from complex types: <extension>

```
<xs:complexType name="PublicationT">
    <xs:sequence>
      <xs:element name="title" type="xs:string" maxOccurs="unbounded"/>
      <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
      <xs:element name="date" type="xs:gYear"/>
    </xs:sequence>
</xs:complexType >

<xs:complexType name="BookPublicationT">
   <xs:complexContent>
      <xs:extension base="PublicationT">
         <xs:sequence>
             <xs:element name="ISBN" type="xs:string"/>
             <xs:element name="publisher" type="xs:string"/>
         </xs:sequence>
         <xs:attribute name="id" type="xs:string"/>
      </xs:extension>
   </xs:complexContent>
```

BookPublicationT has the elements of *PublicationT* + its own

The latter are always added at the end

Complex derived types: restriction

```
<xs:simpleType name="…">
   <xs:restriction base="X">
      …
   </xs:restriction>
</xs:simpleType>
```

X must be a *built-in* or a *simpleType*

```
<xs:complexType name="…">
   <xs:simpleContent>
      <xs:restriction base="Y">
         …
      </xs:restriction>
   </xs:simpleContent>
</xs:complexType>
```

Y must be a *complexType with simpleContent*

```
<xs:complexType name="…">
   <xs:complexContent>
      <xs:restriction base="Z">
         …
      </xs:restriction>
   </xs:simpleContent>
</xs:complexType>
```

Z must be a *complexType with complexContent*

Types derived from simple content types: <restriction>

```
<xs:complexType name="SizeT">
    <xs:simpleContent>
        <xs:extension base="xs:integer">
            <xs:attribute name="system" type="xs:string"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
```

```
<xs:complexType name="SmallSizeT">
    <xs:simpleContent>
        <xs:restriction base="SizeT">
            <xs:minInclusive value="2"/>
            <xs:maxInclusive value="6"/>
            <xs:attribute name="system" type="xs:string"
                                    use="required"/>
        </xs:restriction>
    </xs:simpleContent>
</xs:complexType>
```

Types derived from complex types: <restriction>

```
<xs:complexType name="PublicationT">
        <xs:sequence>
                <xs:element name="Title" type="xs:string" maxOccurs="unbounded"/>
                <xs:element name="Author" type="xs:string" maxOccurs="unbounded"/>
                <xs:element name="Date" type="xs:gYear"/>
        </xs:sequence>
</xs:complexType>

<xs:complexType name= "SingleAuthorPublicationT">
    <xs:complexContent>
        <xs:restriction base="PublicationT">
            <xs:sequence>
                <xs:element name="Title" type="xs:string" maxOccurs="unbounded"/>
                <xs:element name="Author" type="xs:string“ maxOccurs =“1" />
                <xs:element name="Date" type="xs:gYear"/>
            </xs:sequence>
        </xs:restriction>
    </xs:complexContent>
</xs:complexType>
```

SingleAuthorPublicationT has PublicationT’s three elements but one *author* only.
Note that you have to repeat all elements

Types derived from complex types: <restriction>

```
<xs:complexType name="PublicationT">
      <xs:sequence>
            <xs:element name="Title" type="xs:string" maxOccurs="unbounded"/>
            <xs:element name="Author" type="xs:string" minOccurs="0“ maxOccurs =“1”/
            <xs:element name="Date" type="xs:gYear"/>
      </xs:sequence>
</xs:complexType>

<xs:complexType name= "ZeroAuthorPublicationT">
   <xs:complexContent>
      <xs:restriction base="PublicationT">
         <xs:sequence>
            <xs:element name="Title" type="xs:string" maxOccurs="unbounded"/>
            <xs:element name="Date" type="xs:gYear"/>
         </xs:sequence>
      </xs:restriction>
   </xs:complexContent>
</xs:complexType>
```

If the inherited element is optional (minOccurs=0), the derived element can remove it. To attain this, it is enough not to repeat it.

Types derived from complex types: <restriction>

```
<xs:complexType name="PublicationT">
    <xs:sequence>
        <xs:element name="Title" type="xs:string" maxOccurs="unbounded"/>
        <xs:element name="Author" type="xs:string" minOccurs="0“ maxOccurs =“1”/
        <xs:element name="Date" type="xs:gYear"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name= ”SmithPublicationT">
  <xs:complexContent>
    <xs:restriction base="PublicationT">
      <xs:sequence>
        <xs:element name="Title" type="xs:string" maxOccurs="unbounded"/>
        <xs:element name="Author" type="xs:string" minOccurs=”1“ fixed=“Smith”/>
        <xs:element name="Date" type="xs:gYear"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

Substitution mechanisms

- Type substitution xsi:type
  - similar to OO polimorphism
  - one base type can be replaced with any of its derived types

Type substitution. Example

```
<?xml version="1.0"?>
<xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.books.org"
           xmlns="http://www.books.org"
           elementFormDefault="unqualified">
  <xs:complexType name="PublicationType">
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="author" type="xs:string"  maxOccurs="unbounded"/>
      <xs:element name="date" type="xs:year"/>
    </xs:sequence> </xs:complexType>
  <xs:complexType name="BookType">
    <xs:complexContent>
      <xs:extension base="PublicationType">
        <xs:sequence>
          <xs:element name="ISBN" type="xs:string"/>
          <xs:element name="publisher" type="xs:string"/>
        </xs:sequence>
      </xs:extension> </xs:complexContent </xs:complexType>
  <xs:element name="BookStore">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Publication"  maxOccurs="unbounded" type="PublicationType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

PublicationType is the base type

BookType extends *PublicationType*

The *Publication* element is of type *PublicationType*

Type substitution. Example

```
<?xml version="1.0"?>
<xs:schema  … targetNamespace="http://www.books.org" …>
   <xs:complexType name="PublicationType">
      <xs:sequence>
         <xs:element name="title" type="xs:string"/>
         <xs:element name="author" type="xs:string"  maxOccurs="unbound
         <xs:element name="date" type="xs:year"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="BookType">
      <xs:complexContent>
         <xs:extension base="PublicationType">
            <xs:sequence>
               <xs:element name="ISBN" type="xs:string"/>
               <xs:element name="publisher" type="xs:string"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <xs:element name="BookStore">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Publication"
                     maxOccurs="unbounded" type="PublicationType"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

```
<?xml version="1.0"?>
<bk:BookStore  xmlns:bk="http://www.books.org"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation=  "http://www.books.org BookStore.xsd"
>
        <bk:Publication>
                <bk:title>Staying Young Forever</bk:title>
                <bk:author>Karin Jordan, M.D.</bk:author>
                <bk:date>1999</bk:date>
        </bk:Publication>

        <bk:Publication xsi:type="bk:BookType">
                <bk:title>The First and Last Freedom</bk:title>
                <bk:author>J. Krishnamurti</bk:author>
                <bk:date>1954</bk:date>
                <bk:ISBN>0-06-064831-7</bk:ISBN>
                <bk:publisher>Harper Row</bk:publisher>
        </bk:Publication>

</bk:BookStore>
```

The default type is the base type *PublicationType* but any of its derived types can be used

Controlling type derivation (and substitution)

- Three properties of complex types control their derivation:
  - **final**: limits the <u>definition</u> of derived types in schemas
  - **block**: limits the <u>substitution</u> of derived types in instances
  - **abstract**: forces the definition of derived types

Controlling type derivation: **final**

- Purpose: Final types can not be derived
  - Applies to a complex type
- Values: {extension, restriction, #all}

```
<xs:complexType name="PersonT" final="#all">
        ...
</xs:complexType>
```

The designer prevents other people from creating types derived from "PersonT"

- A default value can be assigned at schema level

```
<xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                xmlns:tns="http://example.org/person“
                targetNamespace="http://example.org/person" finalDefault="restriction"
>
```

Controlling type derivation: **block**

- Purpose: elements of type T can restrict the subtypes of T to which they can be instantiated
  - Applies to elements
  - To avoid substitutes/derivates of an element

```
<xs:element  name=“elementName" type=“typeName” block=“?????"/>
```

Example:

```
<xs:element name="Publication”
          maxOccurs="unbounded"  type="PublicationT"
          block=“extension"/>
 </xs:sequence>
```

“Publication” can hold “PublicationT” instances except those whose type is an extension-constructed subtype of “PublicationT”

Controlling type derivation: **block**

- Values:
  - block="substitution": Forbids element substitution
  - block="extension": Forbids type substitution using extension
  - block="restriction": Forbids type substitution using derivation
  - block="#all": Forbids element and type substitution (equivalent to: *block="restriction extension substitution"*)

- A default value can be assigned at schema level

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:tns=http://example.org/person
           targetNamespace="http://example.org/person"
           blockDefault="#all" >
```

Controlling type derivation: **block** Example

```
<xs:complexType name="PublicationType">
    <xs:sequence>
      <xs:element name=“title" type="xs:string"/>
      <xs:element name=“author" type="xs:string"
                                      maxOccurs="unbounded"/>
      <xs:element name=“date" type="xs:year"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="BookType">
    <xs:complexContent>
      <xs:extension base="PublicationType">
        <xs:sequence>
          <xs:element name="ISBN" type="xs:string"/>
          <xs:element name=“publisher" type="xs:string"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:element name=“catalogue">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Publication”
                maxOccurs="unbounded"
                type="PublicationType"
                block="extension"/>
      </xs:sequence>
    </xs:complexType>
```

```
<?xml version="1.0"?>
<bk:bookStore xmlns:bk="http://www.books.org"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance“
>
      <Publication>
            <title>Staying Young Forever</title>
            <author>Karin Jordan, M.D.</author>
            <date>1999</date>
      </Publication>

      <Publication xsi:type="bk:BookType">
            <Title>Illusions </title>
            <author>Richard Bach</author>
            <date>1977</date>
            <ISBN>0-440-34319-4</ISBN>
            <publisher>Dell Publishing Co.</publisher>
      </Publication>
</bk:BookStore>
```

Avoids *Publication* instances to be of a derived type of *PublicationType*

The error is detected when the instance is created

Controlling type derivation: **block** Example

```
<xs:complexType name="PublicationType“ block=“extension”>
      <xs:sequence>
         <xs:element name=“title" type="xs:string"/>
         <xs:element name=“author" type="xs:string"
                                    maxOccurs="unbounded"/>
         <xs:element name=“date" type="xs:year"/>
      </xs:sequence>
</xs:complexType>
<xs:complexType name="BookType">
      <xs:complexContent>
         <xs:extension base="PublicationType">
            <xs:sequence>
               <xs:element name="ISBN" type="xs:string"/>
               <xs:element name=“publisher" type="xs:string"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
</xs:complexType>
<xs:element name=“catalogue">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Publication”
                  maxOccurs="unbounded"
                  type="PublicationType" />
         </xs:sequence>
      </xs:complexType>
</xs:element>
```

```
<?xml version="1.0"?>
<bk:bookStore xmlns:bk="http://www.books.org"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance“
>
      <Publication>
            <title>Staying Young Forever</title>
            <author>Karin Jordan, M.D.</author>
            <date>1999</date>
      </Publication>

      <Publication xsi:type="bk:BookType">
            <Title>Illusions </title>
            <author>Richard Bach</aut
            <date>1977</date>
            <ISBN>0-440-34319-4</ISB
            <publisher>Dell Publishing Co.</publisher>
      </Publication>
</bk:BookStore>
```

informatika fakultatea facultad de informática

Controlling type derivation: **abstract**

- Purpose: Abstract types cannot be instantiated
  - Defines a specification that others will implement. At run time, the abstract type is replaced by one of its derived types
  - Applies to types

Example:

```
<xs:complexType name="TeachingStaffT" abstract="true">
```

- "ReaderT", "LecturerT", "ProfessorT" are defined as derived types from "TeachingStaffT"
  - That means that there CANNOT exist instances of "TeachingStaffT", but there could be of "ReaderT", "LecturerT", "ProfessorT"
- A *taughtBy* element of type "TeachingStaffT" can hold instances of one of its derived types
  - Thus, the designer of *taughtBy* abstracts away from the different "TeachingStaffT" situations that could exist

Contents

DB-like restrictions

- **Nillable**: specifies whether an explicit null value can be assigned to the element
- **<unique>**: defines that an element or an attribute value must be unique within the scope
- **<key>**: specifies an attribute or element value as a key (unique, non-nullable, and always present) within the containing element
- **<keyref>**: specifies that an attribute or element value correspond to those of the specified key or unique element

DB-like restrictions: “Null” value

```
<xs:element name="PersonName">
    <xs:complexType>
        <xs:element name="forename" type="xs:string"/>
        <xs:element name="middle" type="xs:string" nillable="true"/>
        <xs:element name="surname" type="xs:string"/>
    </xs:complexType>
</xs:element>
```

“middle” can contain the *“null”* value

```
<PersonName>
    <forename>John</forename>
    <middle xsi:nil="true"/>
    <surname>Doe</surname>
</PersonName>
```

“null” assignment

John does not have <middle>.
It is not that I forgot to put it

DB-like restrictions: Element uniqueness <unique>

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                    targetNamespace="http://www.books.org”
                    xmlns="http://www.books.org
                    xmlns:bk="http://www.books.org"
                    elementFormDefault="qualified”>
  <xs:element name=“Library">
     <xs:complexType>
        <xs:sequence>
           <xs:element name="Book" maxOccurs="unbounded">
                <xs:complexType>
                   <xs:sequence>
                        <xs:element name=”isbn"
                                type="xs:string" minOccurs =“0"/>
                   <xs:sequence>
               <xs:complexType>
           </xs:element>
        </xs:sequence>
     </xs:complexType>
     <xs:unique name="UNIQ">
        <xs:selector xpath="bk:Book"/>
        <xs:field xpath="bk:isbn"/>
     </xs:unique>
  </xs:element>
</xs:schema>
```

Unlike the key restriction, unique allows for nulls

facultad de informática

DB-like restrictions: Key definition <key>

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.books.org"
           xmlns="http://www.books.org"
           xmlns:bk="http://www.books.org"
           elementFormDefault="qualified">
  <xs:element name="Library">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Book" maxOccurs="unbounded
          <xs:complexType>
            <xs:sequence>
              <xs:element name="isbn"
                   type="xs:string" minOccurs ="1"/>
            <xs:sequence>
          <xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
    <xs:key name="PK">
      <xs:selector xpath="bk:Book"/>
      <xs:field xpath="bk:isbn"/>
    </xs:key>
  </xs:element>
</xs:schema>
```

We want to guarantee that a library doesn't have two books with the same *ISBN* inside a *Library* element

Inside the *Library* element, the following key is defined

- name: *PK*
- inside: *Book*
- on field: *ISBN*

The key element SHOULD be compulsory and MUST not be null i.e. (minOccurs > 0, nillable="false")

Keys always go at the end of the element declaration on which they are defined

DB-like restrictions: <key>. Example

Rule: In the Spanish libraries, there cannot be two books with the same ISBN

```
<xs:schema   xmlns:bk="http://www.books.org“ …>
<xs:element name=“SpanishLibraries">
   <xs:complexType>
        <xs:sequence>
              <xs:element name=“Library“ maxOccurs="unbounded”>
                      <xs:complexType>
                              <xs:sequence>
                                      <xs:element name="Book" maxOccurs="unbounded">
                                            <xs:complexType>
                                                  <xs:sequence>
                                                        <xs:element name=”isbn" type="xs:string" minOccurs =“1"/>
                                                  </xs:sequence>
                                            </xs:complexType>
                                      </xs:element>
                              </xs:sequence>
                      </xs:complexType>
              </xs:element>
         </xs:sequence>
   </xs:complexType>
   <xs:key name="PK">
           <xs:selector xpath=“bk:Library/bk:Book"/>
           <xs:field xpath="bk:isbn"/>
   </xs:key>
</xs:element>
```

DB-like restrictions: <key>. Example

```
<xs:schema targetNamespace="http://www.meeting.org"   xmlns:ns1="http://www.meeting.org" …>
<xs:element name=“Meeting">
        <xs:complexType>
             <xs:sequence>
                 <xs:element name="Participants"  >
                     <xs:complexType>
                         <xs:sequence>
                             <xs:element name=”Participant" minOccurs="0" maxOccurs="unbounded">
                                 <xs:complexType>
                                     <xs:sequence>
                                           <xs:element name="First" type="xs:string"/>
                                           <xs:element name="Last" type="xs:string"/>
                                     </xs:sequence>
                                  </xs:complexType>
                             </xs:element>
                         </xs:sequence>
                     </xs:complexType>
                 </xs:element>
            </xs:sequence>
        </xs:complexType>
        <xs:key name="PK">
              <xs:selector xpath=“ns1:Participants/ns1:Participant"/>
              <xs:field xpath=“ns1:First"/>
              <xs:field xpath=“ns1:Last"/>
        </xs:key>
    </xs:element>
</xs:schema>
```

Rule: In a meeting, there cannot be two people with the same firstname+lastname

A key can be defined on more than one elements

Inside *XPath* always qualify the elements

DB-like restrictions: <keyref>. Example

```
<xs:schema targetNamespace="http://www.books.org"   xmlns:bk="http://www.books.org" ….>
   <xs:element name="Library">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Book" maxOccurs="unbounded">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="isbn" type="xs:string"/>
                     <xs:element name="writer" type="xs:string"/>
                  </xs:sequence>      </xs:complexType>
            </xs:element>
            <xs:element name="Author" maxOccurs="unbounded">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="name" type="xs:string"/>
                  </xs:sequence>    </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
      <xs:key name="PK">
           <xs:selector xpath="bk:Author"/>
           <xs:field xpath="bk:name"/>
      </xs:key>
     <xs:keyref name="FPK" refer="bk:PK">
          <xs:selector xpath="bk:Book"/>
          <xs:field xpath="bk:writer"/>
     </xs:keyref>
   </xs:element>
</xs:schema>
```

Rule: the name of the book writer must be one of the authors registered in the library

Book.writer is a foreign key on *Author.name* in the *Library* context

Contents

© Acknowledgments. Part of these slides have been prepared using R.L. Costello's material

Variability: the issue

- The X from XML stands for “extensible”
  - How to engineer content models for variability?
- Examples:
  - A name can be described by either a string or a compound of firstname and surname
  - A catalogue containing books, magazines, …
  - A list of items of heterogeneous nature (shirt, hat, umbrella)

Variability: Sample problem (1)

- Person names can take two forms:
  - simple-name
  - full-name

```
<simple-name>
  Snoopy
</simple-name>
```

```
<xs:element name="simple-name" type="string32"/>
```

```
<full-name>
  <last>
    Schulz
  </last>
  <first>
    Charles
  </first>
  <middle>
    M
  </middle>
</full-name>
```

```
<xs:element name="full-name">
  <xs:complexType>
    <xs:all>
      <xs:element name="first" type="string32" minOccurs="0"/>
      <xs:element name="middle" type="string32" minOccurs="0"/>
      <xs:element name="last" type="string32"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

Variability: Sample problem (2)

- Catalogue items can be
  - Publications
  - Books
  - Journals

```
<Publication>
      <Title>Staying Young Forever</Title>
      <Author>Karin Jordan, M.D.</Author>
      <Date>1999</Date>
</Publication>

<Book>
      <Title>Illusions</Title>
      <Author>Richard Bach</Author>
       <Date>1977</Date>
       <ISBN>0-440-34319-4</ISBN>
       <Publisher>Dell Publishing Co.</Publisher>
</Book>
```

informatika fakultatea facultad de informática

Variability: The approaches

- Choice structure
- Substitution groups (*element substitution*)
- Subtype mechanism (*type substitution*)

Option 1: choice structure

```
<xs:element name="author">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="name"/>
      <xs:element ref="born"/>
      <xs:element ref="dead" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute ref="id"/>
  </xs:complexType>
</xs:element>
```

```
<xs:group name="name">
  <xs:choice>
    <xs:element ref="simple-name"/
    <xs:element ref="full-name"/>
  </xs:choice>
</xs:group>
```

```
<catalogue  …>
      <author id=“1”>
           <simple-name>William Shakespeare</simple-name>
         …
      </author>
      <author id=“2”>
          <full-name>
              <first>James</first>
              <last>Joyce</last>
          </full-name>
         …
      </author>
</catalogue>
```

Option 1: choice structure

```
<xs:element name="Catalogue">
    <xs:complexType>
        <xs:sequence maxOccurs="unbounded">
            <xs:choice>
                <xs:element ref="Publication"/>
                <xs:element ref="Book"/>
            </xs:choice>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="Publication">…</xs:element>
<xs:element name="Book"> ...</xs:element>
</xs:element>
```

```
<Catalogue>
 <Publication>
        <Title>Staying Young Forever</Title>
        <Author>Karin Jordan, M.D.</Author>
        <Date>1999</Date>
 </Publication>

 <Book>
        <Title>Illusions </Title>
        <Author>Richard Bach</Author>
        <Date>1977</Date>
        <ISBN>0-440-34319-4</ISBN>
        <Publisher>DellPublishing Co.</Publishe
</Book>
</Catalogue>
```

Option 2: substitutionGroup

- Group of elements that can be used wherever the base element shows
  - Substitution relation is transitive, but not commutative
  - Substitute elements have to be global
  - The type of substitute elements has to be derived from the type of the element to be substitued

Example:

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

<xs:element name="casa" substitutionGroup="house" type="xs:string"/>

<xs:element name="etxe" substitutionGroup="house" type="xs:string"/>

<xs:element name="maison" substitutionGroup="house" type="xs:string"/>
```

Wherever the <house> element appears, any of its substitutes can appear

Option 2: substitutionGroup

Example

```
<xs:element name="name" type = “xs:anyType” abstract= “true”/>
```

```
<xs:element name="simple-name" type="string32"
  substitutionGroup="name"/>
```

```
<xs:element name="full-name" substitutionGroup="name">
  <xs:complexType>
    <xs:all>
      <xs:element name="first" type="string32" minOccurs="0"/>
      <xs:element name="middle" type="string32" minOccurs="0"/>
      <xs:element name="last" type="string32"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

```
<xs:element name=“author”>
       …
       < xs:element ref=“name” />
         …
 </xs:element>
```

```
<catalogue …>
      <author>
            <simple-name>William Shakespeare</simple-name>
         …
      </author>
      <author>
          <full-name>
             <first>James</first>
             <last>Joyce</last>
          </full-name>
         …
      </author>
</catalogue>
```

Option 2: substitutionGroup

Example

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema” …  >
   <xs:complexType name="PublicationType">
      <xs:sequence>
         <xs:element name="Title" type="xs:string"/>
         <xs:element name="Author" type="xs:string"  maxOccurs="unbounded"/>
         <xs:element name="Date" type="xs:year"/>
      </xs:sequence> </xs:complexType>
   <xs:complexType name="BookType">
      <xs:complexContent>
         <xs:extension base="PublicationType">
            <xs:sequence>
               <xs:element name="ISBN" type="xs:string"/>
               <xs:element name="Publisher" type="xs:string"/>
            </xs:sequence>
         </xs:extension> </xs:complexContent> </xs:complexType>
  <xs:complexType name=”MagazineType">
      <xs:complexContent>
           <xs:extension base="PublicationType”> …</xs:extension> …
  </xs:complexType>
  <xs:element name=“catalogue”>
     <xs:complexType>
         <xs:sequence> <xs:element ref=”publication” maxOccurs=“unbounded”/> </xs:sequence>
    </xs:complexType> </xs:element>
   <xs:element name="publication" type="PublicationType"/>
   <xs:element name="book" substitutionGroup="publication" type="BookType"/>
   <xs:element name="journal" substitutionGroup="publication" type=”MagazineType”/>
</xs:schema>
```

facultad de informática

Option 2: substitutionGroup

Example

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" …  >
   <xs:complexType name="PublicationType">
      <xs:sequence>
         <xs:element name="Title" type="xs:string"/>
         <xs:element name="Author" type="xs:string"  maxOccurs="unbounded"/>
         <xs:element name="Date" type="xs:year"/>
      </xs:sequence> </xs:complexType>
  <xs:complexType name="BookType"> … </xs:complexType>
  <xs:complexType name=”MagazineType"> … </xs:complexType>
  <xs:complexType name=”RegistratedBookType">
      <xs:complexContent>
         <xs:extension base=”BookType”>
              <xs:attribute name=”signature" type="xs:string"/>
          </xs:extension> </xs:complexContent>
  </xs:complexType>
  <xs:element name=“catalogue”>
     <xs:complexType>
         <xs:sequence> <xs:element ref=”publication” maxOccurs=“unbounded”/> </xs:sequence>
    </xs:complexType> </xs:element>
   <xs:element name=“publication" type=”PublicationType"/>
   <xs:element name=“book" substitutionGroup=“publication" type="BookType"/>
   <xs:element name=“journal" substitutionGroup=“publication" type=”MagazineType”/>
   <xs:element name=“regbook" substitutionGroup=“book" type=”RegistratedBookType"/>
</xs:schema>
```

Option 2: substitutionGroup
Attributes for substitution groups

- *abstract*, to define abstract elements inside the substitution group

```
<xs:element name=“house" type="xs:string“ abstract =“true”/>
```

  - There cannot exist documents with “house”. We can only use one of the substitutes

- *block*, to avoid the use of substitutes

```
<xs:element name=“house” type="xs:string“ block =“substitution” />
```

  - There cannot exist documents with substitutes for the *“house”* element

Option 3: Subtype mechanisms

- Type substitution (*xsi:type*)
  - similar to OO polimorphism
  - one base type can be replaced with any of its derived types

Type substitution. Example

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.books.org"
           xmlns="http://www.books.org"
           elementFormDefault="unqualified">
  <xs:complexType name="PublicationType">
    <xs:sequence>
      <xs:element name="Title" type="xs:string"/>
      <xs:element name="Author" type="xs:string"  maxOccurs="unbounded"/>
      <xs:element name="Date" type="xs:year"/>
    </xs:sequence> </xs:complexType>
  <xs:complexType name="BookType">
    <xs:complexContent>
      <xs:extension base="PublicationType">
        <xs:sequence>
          <xs:element name="ISBN" type="xs:string"/>
          <xs:element name="Publisher" type="xs:string"/>
        </xs:sequence>
      </xs:extension> </xs:complexContent </xs:complexType>
  <xs:element name="BookStore">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Publication"  maxOccurs="unbounded" type="PublicationType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

PublicationType is the base type

BookType extends *PublicationType*

The *Publication* element is of type *PublicationType*

Type substitution. Example

```
<?xml version="1.0"?>
<xs:schema  …>
   <xs:complexType name="PublicationType">
      <xs:sequence>
        <xs:element name="Title" type="xs:string"/>
        <xs:element name="Author" type="xs:string"  maxOccurs="unbounded"/>
        <xs:element name="Date" type="xs:year"/>
     </xs:sequence>
   </xs:complexType>
   <xs:complexType name="BookType">
      <xs:complexContent>
        <xs:extension base="PublicationType">
           <xs:sequence>
              <xs:element name="ISBN" type="xs:string"/>
              <xs:element name="Publisher" type="xs:string"/>
           </xs:sequence>
        </xs:extension>
     </xs:complexContent>
   </xs:complexType>
   <xs:element name="BookStore">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Publication“
                  maxOccurs="unbounded" type="PublicationType"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

```
<?xml version="1.0"?>
<bk:BookStore  xmlns:bk="http://www.books.org"
       xmlns:xsi=“…"
       xsi:schemaLocation=  "http://www.books.org
                                BookStore.xsd">
       <Publication>
              <Title>Staying Young Forever</Title>
              <Author>Karin Jordan, M.D.</Author>
              <Date>1999</Date>
       </Publication>

       <Publication xsi:type="bk:BookType">
              <Title>The First and Last Freedom</Title>
              <Author>J. Krishnamurti</Author>
              <Date>1954</Date>
              <ISBN>0-06-064831-7</ISBN>
              <Publisher>Harper Row</Publisher>
       </Publication>
```

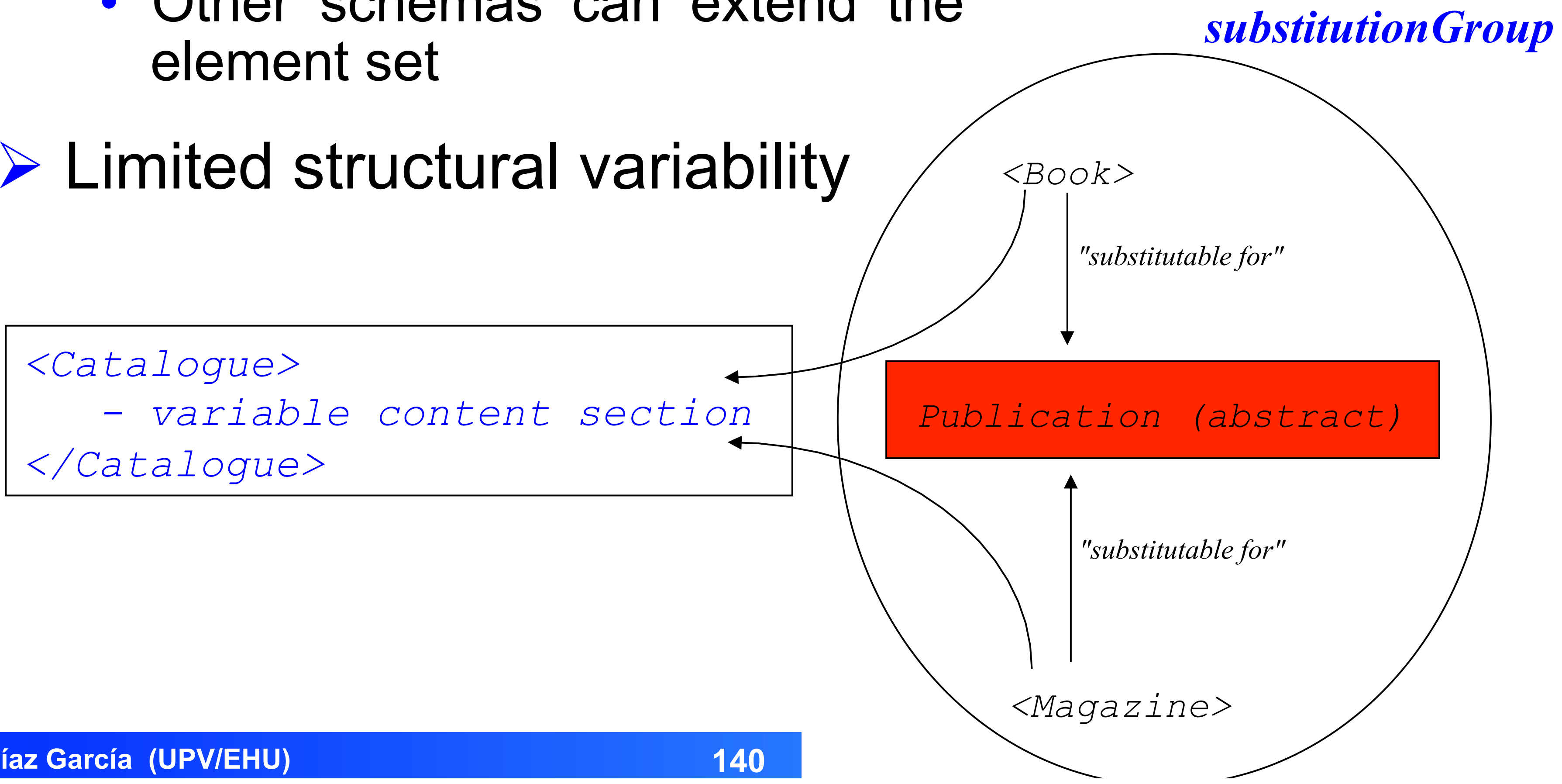
The default type is the base type *PublicationType* but any of its derived types can be used

Recap: using a choice element

- No semantic coherence
  - Items are not type related
  - Catalogue can be of books, bikes, breaks
- Non extensible (for read-only schemas)
  - You can not add new item types to the catalogue

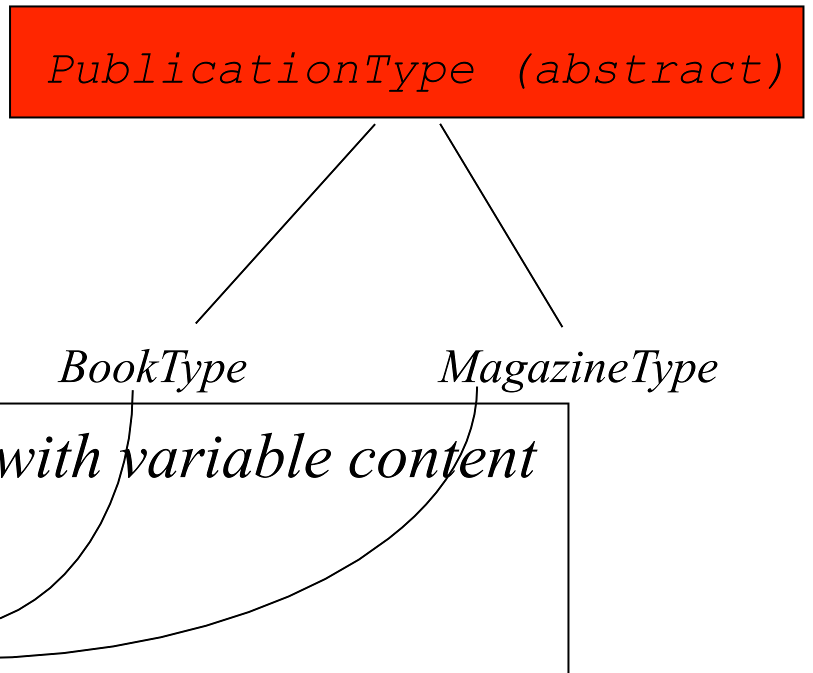
Recap: Abstract element and *element substitution*

- Semantic cohesion
- Extensibility
  - Other schemas can extend the element set
- Limited structural variability



Recap: Abstract type and *type substitution*

- Semantic cohesion
- Extensibility
  - Other schemas can extend the element set
- Limited structural variability



Only **`<Publication>`** *elements but with variable content*

```
<Catalogue>
   <Publication xsi:type="…">
      - variable content section
   </Publication>
</Catalogue>
```

Contents

- Motivation
- Schema: basics
- Schemas and documents
  - Schema is a document
  - Associating a schema to a document
- Schema definition
  - Attributes, Elements, Types
  - Database-like restrictions (Null value, key, foreign key)
- Schema variability
- Handling Schema complexity
  - **<include>**, **<import>**, **<redefine>**
- Schema extensibility
- UML and XML Schema

Schema complexity

A schema can be very complex

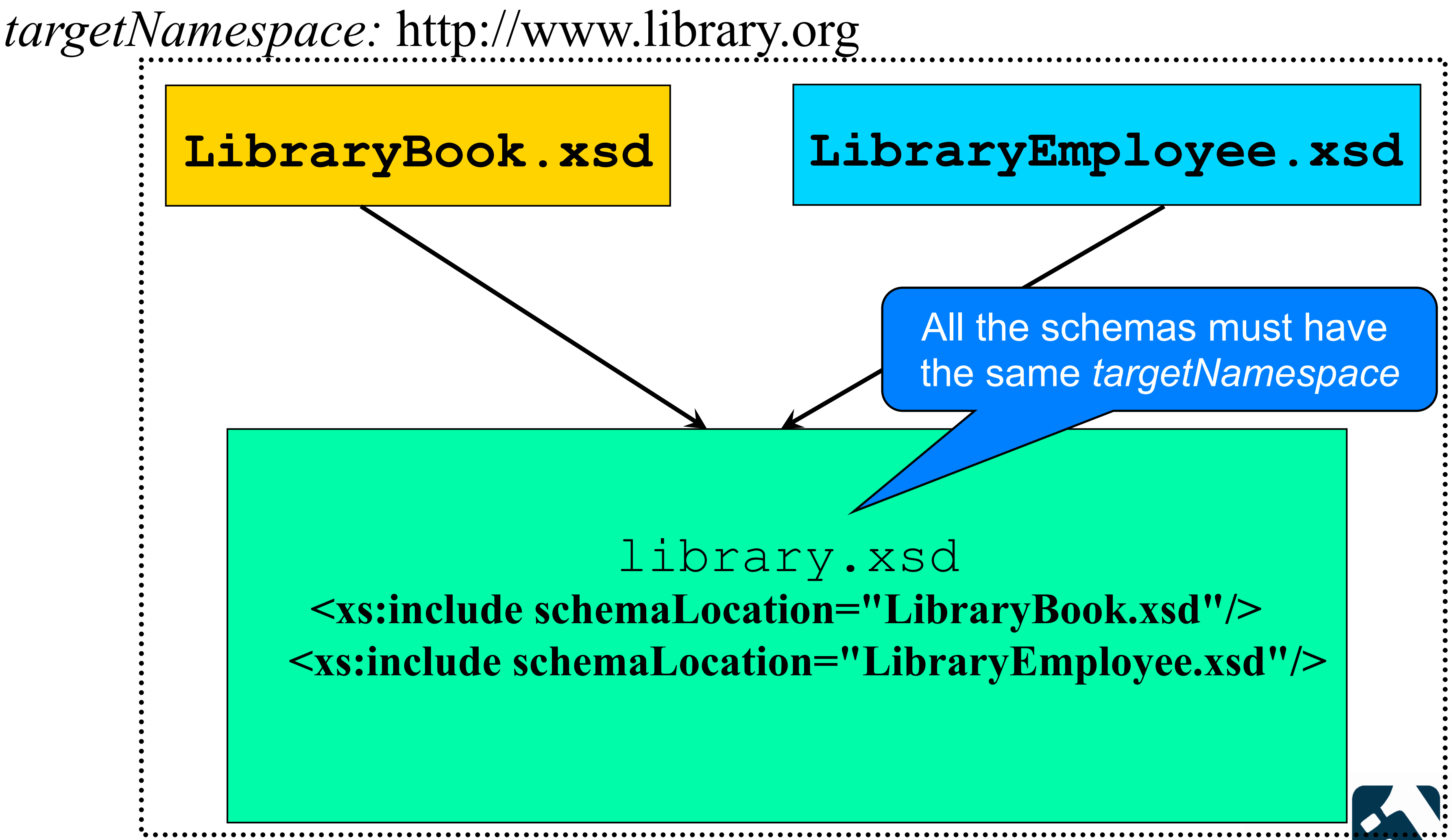
- Are there modularization mechanisms?
  - use of different schemas
- Are there reuse mechanisms?

informatika fakultatea facultad de informática

Different schemas.
Perspective from the schema document

- A schema can be created using other schemas
- Two options:
  - **<include>**: one namespace spreads among several ".xsd"
  - **<import>**: one namespace that uses other namespaces

<include>


targetNamespace: http://www.library.org
LibraryBook.xsd
LibraryEmployee.xsd
All the schemas must have the same *targetNamespace*
library.xsd

```
<xs:include schemaLocation="LibraryBook.xsd"/>
<xs:include schemaLocation="LibraryEmployee.xsd"/>
```

<include> : Schema perspective

LibraryBook.xsd

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

   <xs:complexType name="BookT">
       <xs:sequence>
               <xs:element name="title" type="xs:string"/>
               <xs:element name="author" type="xs:string" />
               <xs:element name="date" type="xs:string"/>
               <xs:element name="ISBN" type="xs:string"/>
               <xs:element name="publisher" type="xs:string"/>
       </xs:sequence>
   </xs:complexType>
   <xs:element name="Book" type="BookT" />
</xs:schema>
```

LibraryEmployee.xsd

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

   <xs:complexType name="EmployeeT">
       <xs:sequence>
               <xs:element name="tname" type="xs:string"/>
               <xs:element name="SSN" type="xs:string" />
       </xs:sequence>
   </xs:complexType>
   <xs:element name="Employee" type="EmployeeT" />
</xs:schema>
```

<include> : Schema perspective

Library.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.library.org"
           xmlns="http://www.library.org"
           elementFormDefault="qualified">
  <xs:include schemaLocation="LibraryBook.xsd"/>
  <xs:include schemaLocation="LibraryEmployee.xsd"/>
  <xs:element name="Library">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Books">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="Book" maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Employees">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="Employee" maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

targetNamespace
All the included schemas must have the same *targetNamespace*

<include>
Has the same effect as if the schema was specified in this file. It can be overwritten

Use of included elements
An included element/attribute is indicated by reference

<include> : Instance perspective

```xml
<?xml version="1.0"?>
<Library xmlns="http://www.library.org”>
    <Books>
        <Book>
                <Title>My Life and Times</Title>
                <Author>Paul McCartney</Author>
                <Date>1998</Date>
                <ISBN>1-56592-235-2</ISBN>
                <Publisher>Macmillan Publishing</Publisher>
        </Book>
        <Book>
                <Title>The First and Last Freedom</Title>
                <Author>J. Krishnamurti</Author>
                <Date>1954</Date>
                <ISBN>0-06-064831-7</ISBN>
                <Publisher>Harper &amp; Row</Publisher>
        </Book>
    </Books>
    <Employees>
        <Employee>
            <name>John Doe</name>
            <SSN>123-45-6789</SSN>
        </Employee>
  </Employees>
</Library>
```

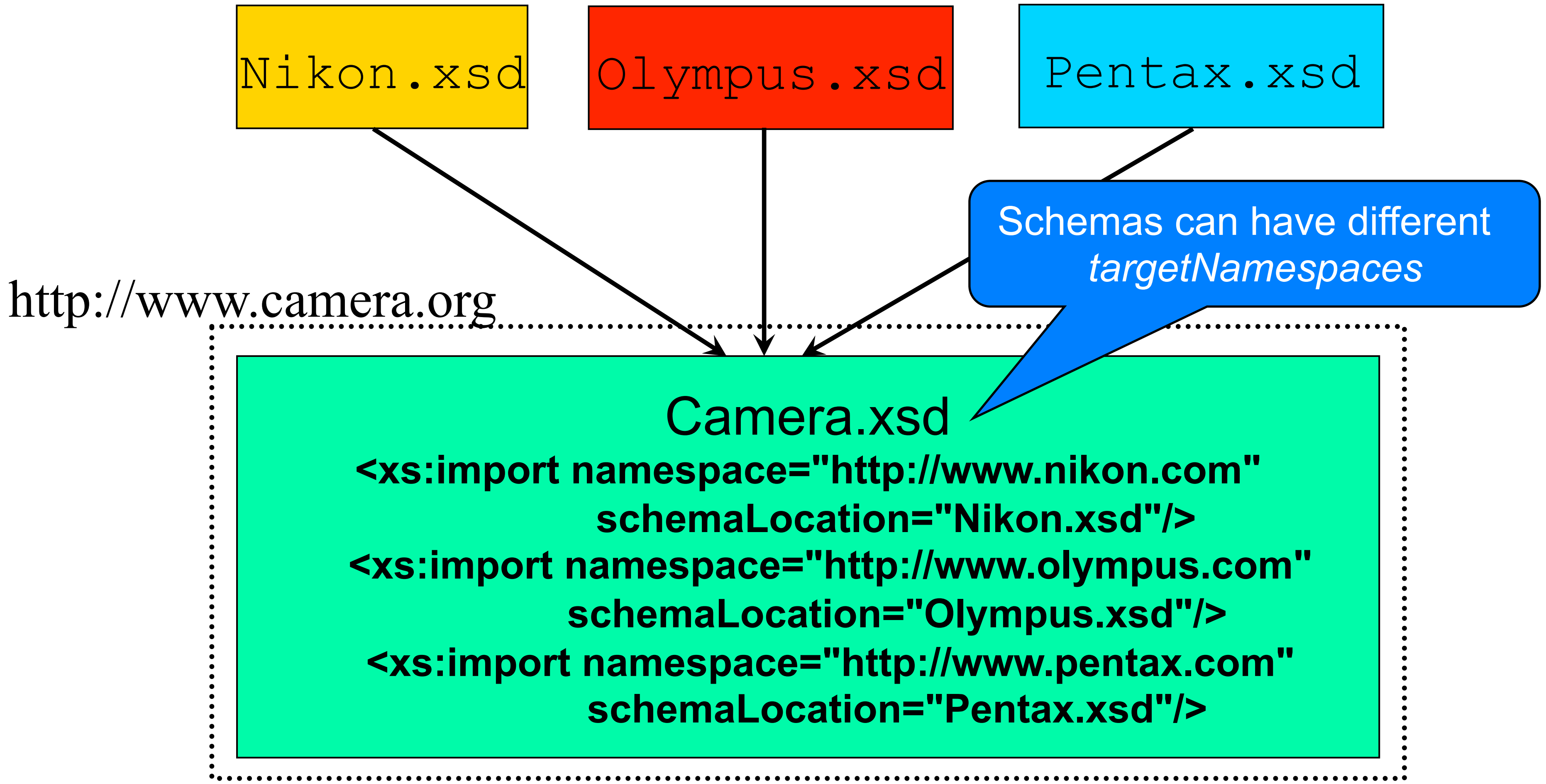
The instance document indicates the *www.library.org* vocabulary

To sum up, *<include>* allows a modular approach when defining the same namespace

informatika fakultatea

facultad de informática

<import>



```
<xs:import namespace="http://www.nikon.com"
           schemaLocation="Nikon.xsd"/>
<xs:import namespace="http://www.olympus.com"
           schemaLocation="Olympus.xsd"/>
<xs:import namespace="http://www.pentax.com"
           schemaLocation="Pentax.xsd"/>
```

Nikon.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.nikon.com"
           xmlns="http://www.nikon.com"
           elementFormDefault="qualified">
  <xs:complexType name="body_type">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Olympus.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.olympus.com"
           xmlns="http://www.olympus.com"
           elementFormDefault="qualified">
  <xs:complexType name="lens_type">
    <xs:sequence>
      <xs:element name="zoom" type="xs:string"/>
      <xs:element name="f-stop" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Pentax.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.pentax.com"
           xmlns="http://www.pentax.com"
           elementFormDefault="qualified">
  <xs:complexType name="manual_adapter_type">
    <xs:sequence>
      <xs:element name="speed" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Schemas have different *targetNamespaces*

<import>: Schema perspective

Camera.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  targetNamespace="http://www.camera.org"
                  xmlns="http://www.camera.org"
                  xmlns:nikon="http://www.nikon.com"
                  xmlns:olympus="http://www.olympus.com"
                  xmlns:pentax="http://www.pentax.com"
                  elementFormDefault="qualified">
   <xs:import namespace="http://www.nikon.com"
            schemaLocation="Nikon.xsd"/>
   <xs:import namespace="http://www.olympus.com"
            schemaLocation="Olympus.xsd"/>
   <xs:import namespace="http://www.pentax.com"
            schemaLocation="Pentax.xsd"/>

   <xs:element name="camera">
      <xs:complexType>
          <xs:sequence>
             <xs:element name="body" type="nikon:body_type"/>
             <xs:element name="lens" type="olympus:lens_type"/>
             <xs:element name="manual_adapter" type="pentax:manual_adapter_type"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>

<xs:schema>
```

targetNamespace
The imported schemas DON'T have the same *targetNamespace*

<import>
Imports the definitions. As they have different *namespaces*, we have to indicate where each of them is located

Including imported elements
Using qualifiers

<import>: Instance perspective

```
<?xml version="1.0"?>
<c:camera xmlns:c="http://www.camera.org"
          xmlns:nikon="http://www.nikon.com"
          xmlns:olympus="http://www.olympus.com"
          xmlns:pentax="http://www.pentax.com"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation=
               "http://www.camera.org
                Camera.xsd">
   <c:body>
      <nikon:name>Ergonomically designed casing for easy handling</nikon:name>
   </c:body>
   <c:lens>
      <olympus:zoom>300mm</olympus:zoom>
      <olympus:f-stop>1.2</olympus:f-stop>
   </c:lens>
   <c:manual_adapter>
      <pentax:speed>1/10,000 sec to 100 sec</pentax:speed>
   </c:manual_adapter>
</c:camera>
```

The instance document must specify ALL the *namespaces*

However, it is enough to locate the “importing” schema, as it already has the other locations

To sum up. <*import*> allows using other vocabularies when defining a vocabulary

<redefine>

```
<xs:complexType name="BookType">
    <xs:sequence>
        <xs:element name="Title" type="xs:string"/>
        <xs:element name="Author" type="xs:string"/>
        <xs:element name="Date" type="xs:string"/>
        <xs:element name="ISBN" type="xs:string"/>
        <xs:element name="Publisher" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
```

LibraryBook.xsd

```
<xs:redefine schemaLocation="LibraryBook.xsd">
    <xs:complexType name="BookType">
        <xs:complexContent>
            <xs:extension base="BookType">
                <xs:sequence>
                    <xs:element name="Summary" type="xs:string
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:redefine>
```

Library.xsd

<redefine>
The same as *<include>* but it also allows redefining any "component"

- *simpleType,*
- *complexType,*
- *attributeGroup*
- *group*

Any reference to *BookType* both in *Library.xsd* and *LibraryBook.xsd* will be to the new redefined type

Contents

© Acknowledgments. Part of these slides have been prepared using R.L. Costello’s material

tad de
nática

Roles

- Global schema author
  - (e.g. a standarisation committee)
- Local schema author
  - (e.g. a local authority, the company's DB administrator)
- Document instance author
  - (e.g. a programmer)
- Notice that
  - the author of the schema (.xsd) and of the instance (.xml) are usually different
  - Global schemas tend to be adapted for local use

Extensibility points: **<any>**

- They allow documents to contain additional elements that are not declared in the main XML schema

- Role "schema author"
  - indicates where the schema can be extended
- Role "document author"
  - can provide new elements not declared in the schema

Role “schema author” where do I allow extending the schema?

```
targetNamespace="http://www.BookRetailers.org">
<xs:complexType name="BookT">
        <xs:sequence>
            <xs:element name="Title" type="xs:string"/>
            <xs:element name="Author" type="xs:string"/>
            <xs:element name="Date" type="xs:string"/>
            <xs:element name="ISBN" type="xs:string"/>
            <xs:element name="Publisher" type="xs:string"/>
            <xs:any  namespace = “##any” minOccurs="0"/>
        </xs:sequence>
        <xs:anyAttribute namespace = “##any”/>
</xs:complexType>
<xs:element name=“BookSeller”>
    <xs:complexType>
        <xs:sequence>
            <xs:element name=“Book” type=“BookT” maxOccurs=“unbounded”/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

The author of schema *Book* allows adding new elements

The author of schema *Book* allows adding new attributes

facultad de informática

Role “document author” what do I want to extend the schema with?

- He defines a NEW schema with his own elements/attributes

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                     targetNamespace="http://www.myOwnSchema.org">
<element name =“Reviewer”>
       <complexType>
             <sequence>
                 <element name="name">
                     <complexType>
                         <sequence>
                             <element name="First" type="xs:string"/>
                             <element name="Last" type="xs:string"/>
                         </sequence>
                     </complexType>
                 </element>
            </sequence>
       <complexType>
 </element>
<attribute name=”recommendable" type="xs:boolean"/>
</xs:schema>
```

Thus:

1. creates his own vocabulary: *www.myOwnSchema.org*

At the time the document is defined…

```
<?xml version="1.0"?>
<BookSeller xmlns="http://www.BookRetailers.org"
            xmlns:own="http://www.myOwnSchema.org" />
    <Book own:recommendable=”false”>
        <Title>My Life and Times</Title>
        <Author>Paul McCartney</Author>
        <Date>1998</Date>
        <ISBN>1-56592-235-2</ISBN>
        <Publisher>McMillin Publishing</Publisher>
        <Reviewer xmlns="http://www.myOwnSchema.org">
          <name>
            <First>Roger</First>
            <Last>Costello</Last>
          </name>
        </Reviewer>
    </Book>
    <Book own:recommendable=”true">
        <Title>Illusions: The Adventures of a Reluctant M
        <Author>Richard Bach</Author>
        <Date>1977</Date>
        <ISBN>0-440-34319-4</ISBN>
        <Publisher>Dell Publishing Co.</Publisher>
    </Book>
</BookSeller>
```

Combine both vocabularies

creates the instance document using both vocabularies, knowing that elements from *repository* can be used wherever *Book* allows it

informatika fakultatea facultad de informática

Restricting the vocabularies that can extend the schema

<anyAttribute namespace="##any"/>

allows any new attribute. Default value

<anyAttribute namespace="http://www.somewhere.com"/>

allows new attributes, only if they come from the specified vocabulary
Note: more than one can be specified, separated by spaces

<anyAttribute namespace="##targetNamespace"/>

allows attributes from the vocabulary that is being defined (*targetNamespace*)

<anyAttribute namespace="##other"/>

allows the instances to have new attributes, providing that those attributes are NOT defined in the *targetNamespace*

<anyAttribute namespace="##local"/>

allows any attribute that is NOT defined in any vocabulary (i.e. with no namespace)

* exactly the same for ***<any>***

Contents

© Acknowledgments. Part of these slides have been prepared using R.L. Costello's material

UML and *XML Schema*

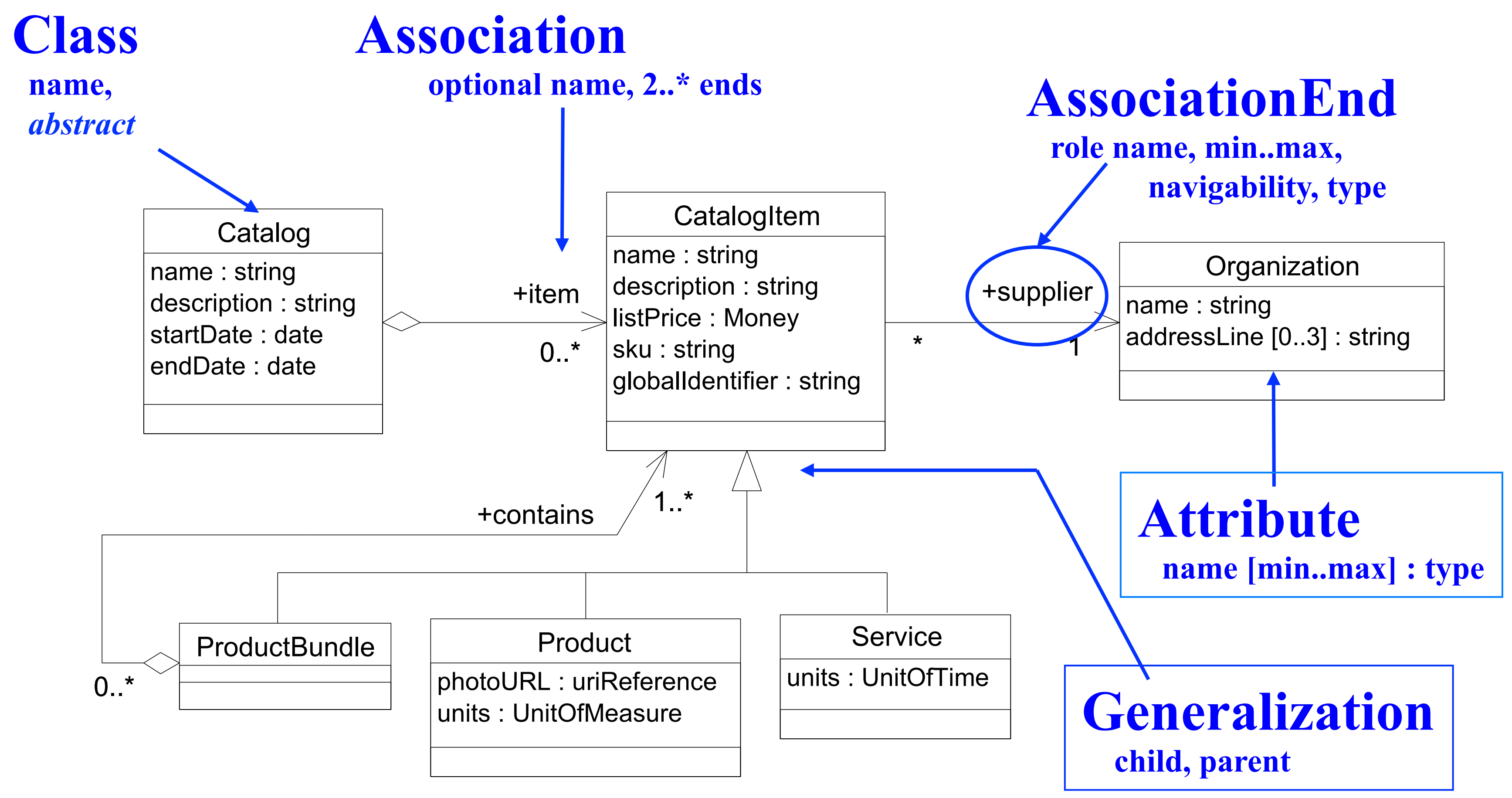
- *UML* is visual → design tool
- *XML Schema* is code → implementation tool

- Use UML to design document schemas

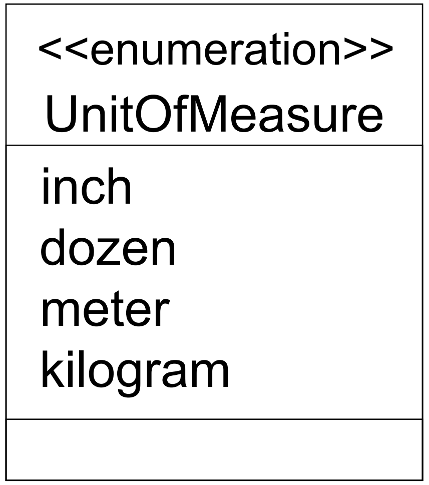
- There is controversy about the correspondence between the primitives of UML and *XML Schema*

- Here some examples taken from **David Carlson, Ontogenics Corp.** are shown

Example

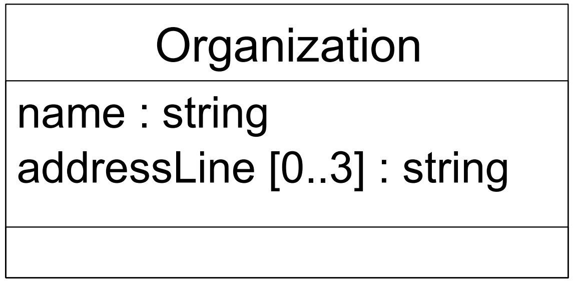

Class
name,
abstract
Association
optional name, 2..* ends
AssociationEnd
role name, min..max,
navigability, type
Catalog
name : string
description : string
startDate : date
endDate : date
+item
0..*
CatalogItem
name : string
description : string
listPrice : Money
sku : string
globalIdentifier : string
+supplier
*
1
Organization
name : string
addressLine [0..3] : string
Attribute
name [min..max] : type
+contains
1..*
0..*
ProductBundle
Product
photoURL : uriReference
units : UnitOfMeasure
Service
units : UnitOfTime
Generalization
child, parent

Enumeration: simple type with restriction



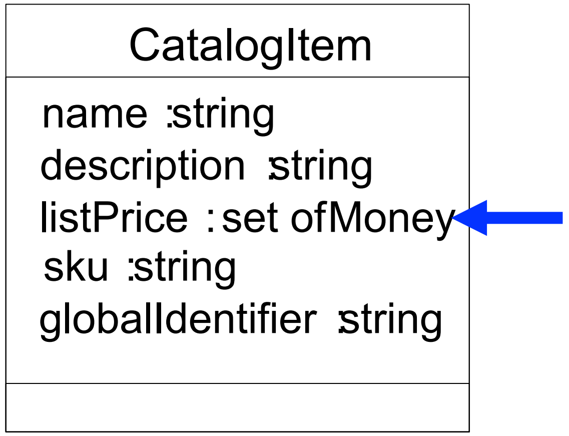
```
<xs:simpleType name="unitOfMeasure">
    <xs:restriction base="xs:string">
        <xs:enumeration value=“inch"/>
        <xs:enumeration value="dozen"/>
        <xs:enumeration value="meter"/>
        <xs:enumeration value="kilogram"/>
    </xs:restriction>
</xs:simpleType>
```

Class: <sequence> type



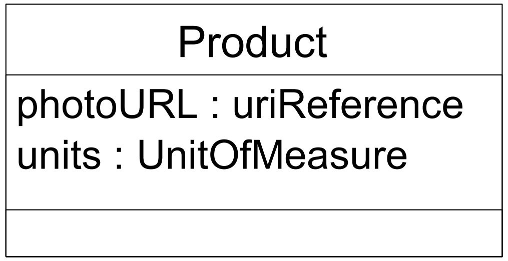
```
<xs:element name="Organization” type=”OrganizationT"/>
<xs:complexType name="OrganizationT">
    <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="addressLine" type="xs:string"
                minOccurs="0" maxOccurs="3” />
    </xs:sequence>
</xs:complexType>
```

Class: <all> type


CatalogItem
name :string
description :string
listPrice : set ofMoney
sku :string
globalIdentifier :string

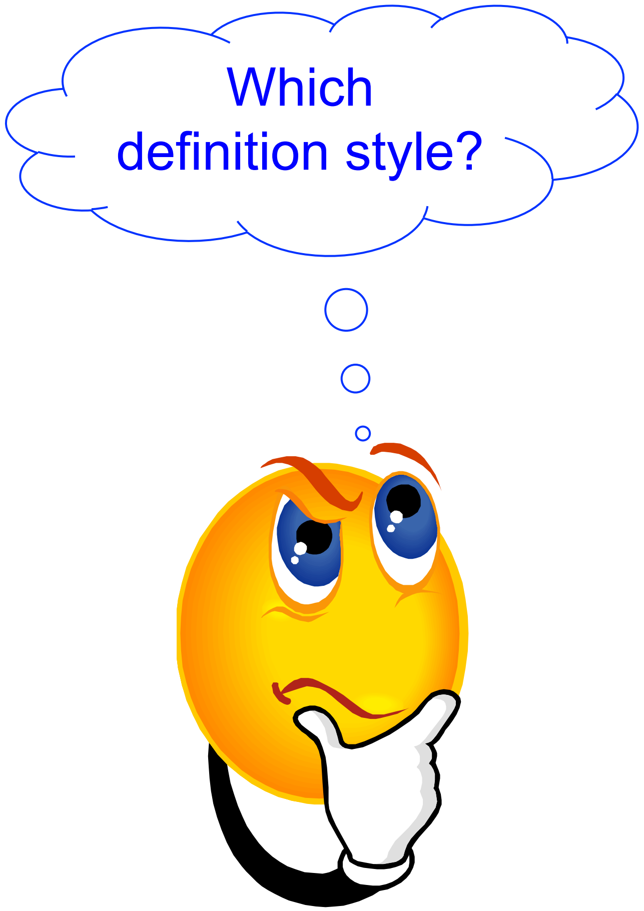
```
<xs:element name="CatalogItem" type="CatalogItemT"/>
<xs:complexType name="CatalogItemT">
  <xs:all>
     <xs:element name="name" type="xs:string"/>
     <xs:element name="description" type="xs:string"/>
     <xs:element name="listPrice">
       <xs:complexType>
         <xs:sequence>
            <xs:element ref="Money" maxOccurs="unbounded"/>
         </xs:sequence>
       </xs:complexType>
     </xs:element>
     <xs:element name="sku" type="xs:string"/>
     <xs:element name="globalIdentifier" type="xs:string"/>
  </xs:all>
</xs:complexType>
```

Class: type with attribute



```
<xs:element name="Product" type="ProductT"/>
<xs:complexType name = "ProductT>
    <xs:attribute ref="photoURL" use="required"/>
    <xs:attribute ref="units" use="required"/>
</xs:complexType>
```

Attribute vs. Element



```
<karta>
    <item portzio-tamaina="250 mL">
        <izena>Arabar Errioxa</izena>
    </item>
    <item portzio-tamaina="500 gr">
        <izena>tortila pintxoa</izena>
    </item>
</karta>

<karta>
    <item portzio-tamaina="250" portzio-unitatea="mL">
        <izena>Arabar Errioxa</izena>
    </item>
    <item portzio-tamaina="500" portzio-unitatea="gr">
        <izena>tortila pintxoa</izena>
    </item>
</karta>

<karta>
    <item>
        <portzioa unitatea="mL">250</portzioa>
        <izena>Arabar Errioxa</izena>
    </item>
    <item>
        <portzioa unitatea="gr">500</portzioa>
        <izena>tortila pintxoa</izena>
    </item>
</karta>
```

Attribute vs. Element. Guidelines

- Guide: syntactic restrictions
  - Element
    - has structure
    - can be repeated
    - is not “normalized”
  - Attribute
    - is atomic
    - cannot be repeated
    - is usually “normalized”

Attribute vs. Element. Guidelines (2)

- Guide of data vs. meta-data
  - Element
    - data
    - nuclear part of the document
  - Attribute
    - meta-data
    - helps to understand, process, classify the document (e.g. author, creationDate, identifier..)

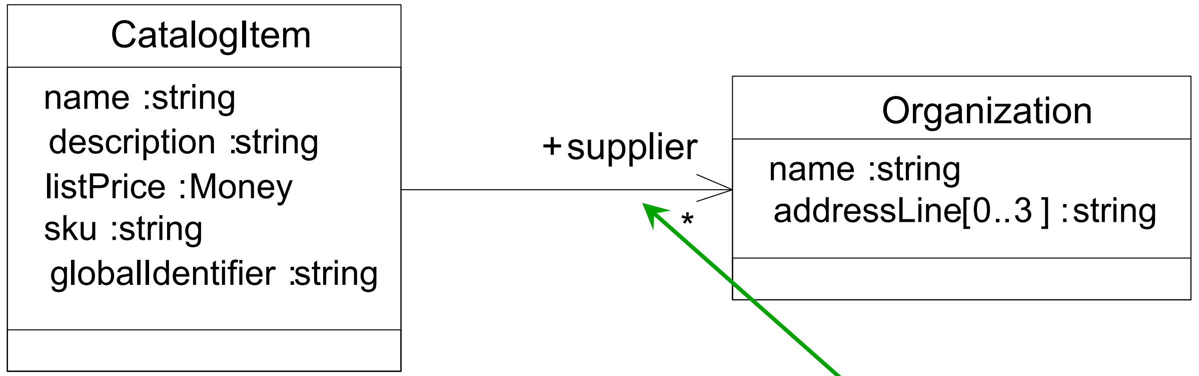
Attribute vs. Element. Guidelines (3)

- Guide: Target audience
  - Element
    - If thought for person consumption
  - Attribute
    - If thought for processor consumption (e.g. URL, images, IDs, …)

Attribute vs. Element. Guidelines (4)

- Guide: existence of qualifiers
  - Element
    - If there are other elements that qualify or describe it
  - Attribute
    - If it is a qualifier or descriptor (e.g. measure)

Association: element

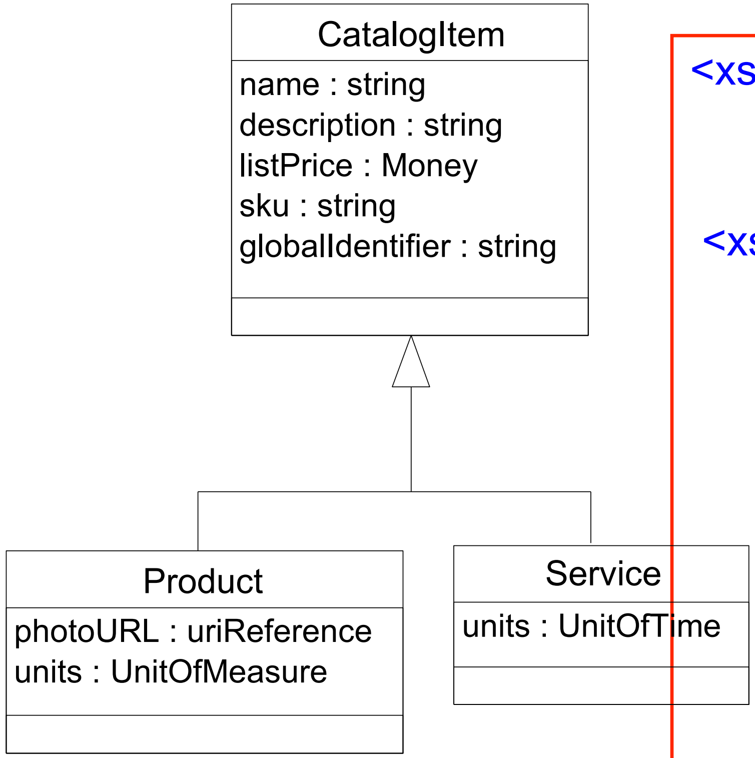


```
<xs:complexType name="CatalogItemT" >
  <xs:all>
     . . .
    <xs:element name="supplier">
        <xs:complexType>
           <xs:sequence>
              <xs:element ref="Organization“ maxOccurs=“unbounded” minOccurs=“0” />
           </xs:sequence>
        </xs:complexType>
    </xs:element>
  </xs:all>
</xs:complexType>
<xs:complexType name=“OrganizationT">  … </xs:complexType>
<xs:element name="Organization“ type=“OrganizationT>
<xs:element name=“CatalogItem“ type=“CatalogItemT>
```

Guarantees the cardinality of the association

Specialization → substitutionGroup

```
<xs:element name="CatalogItem" type="CatalogItemT" />
<xs:complexType name="CatalogItemT"> …</xs:complexType>
```



```
<xs:element name="Product" type="ProductT"
                                    substitutionGroup="CatalogItem" />

 <xs:complexType name="ProductT">
     <xs:complexContent>
          <xs:extension base="CatalogItemT">
               <xs:all>
                    <xs:element name="photoURL"
                                   type="xs:uriReference"/>
                    <xs:element name="units"
                                   type="unitsOfMeasure"/>
               </xs:all>
          </xs:extension>
     </xs:complexContent>
 </xs:complexType>
<xs:element name="Service" type="ServiceT"
                                         substitutionGroup="CatalogItem" />
<xs:complexType name="ServiceT"> …</xs:complexType>
```