XML Schema

DTD limitations

1. Not written in XML syntax, DTD has its own syntax. So it is hard to learn.
2. Precise number of element repetitions can’t be achieved.
3. XML document can reference only 1 DTD.
4. Do not support namespaces

DTD Disadvantages (Cont'd)

5. no constraints on character data:
   - **Desired:**

     ```
     <quantity-Kg>55</quantity-Kg>
     ```

     Is valid
   - **Not Desired**

     ```
     <quantity-Kg>hello</quantity-Kg>
     ```

     But it is valid in DTD.

6. too simple attribute value models

XML Schema

- XML schema is an XML based alternative to DTD
  - Use XML syntax easy to learn – extensible.
  - Support namespaces.
  - Can ensure proper element content, it supports non textual data types. ”Integer, decimal, … etc ”
    - **Not Desired**

```
<quantity-Kg>hello</quantity-Kg>
```

We can make it not valid in Schema.

- XML document that conforms to an XML schema is said to be “*schema valid* ”

XML Schema (Cont'd)

- XML Schemas are a tremendous advancement over DTDs:
  - It has 44 Enhanced data-types .
  - You can create your **own** data-types.
  - Can express sets, i.e., can define the child elements to occur in any order to be demonstrated.
  - It is Object-Oriented

    Can extend or restrict a type.

schema Syntax

- XML document with root element schema

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

How to define XML Element in schema?

- **General Form:**

  of defining XML element in schema:

```
<xs:element name="??" Optional_Atribuites???/>
```

  - **xs:element:** used to define "XML element".
  - **Optional Attributes** : include "type, default, fixed, final, minOccurs, maxOccurs, …etc"

How to define XML Element in schema?

- **Type:** Define element type and it can be:
  - **Built-in** type “decimal, string, Integer, … etc”
  - Or “**User** defined”.

- **Using Built-in Types:**

- In Schema:

```
<xs:element name="Price" type="xs:decimal"/>
```

- In XML:

```
<Price>130</Price>
```

Built-in Simple Types

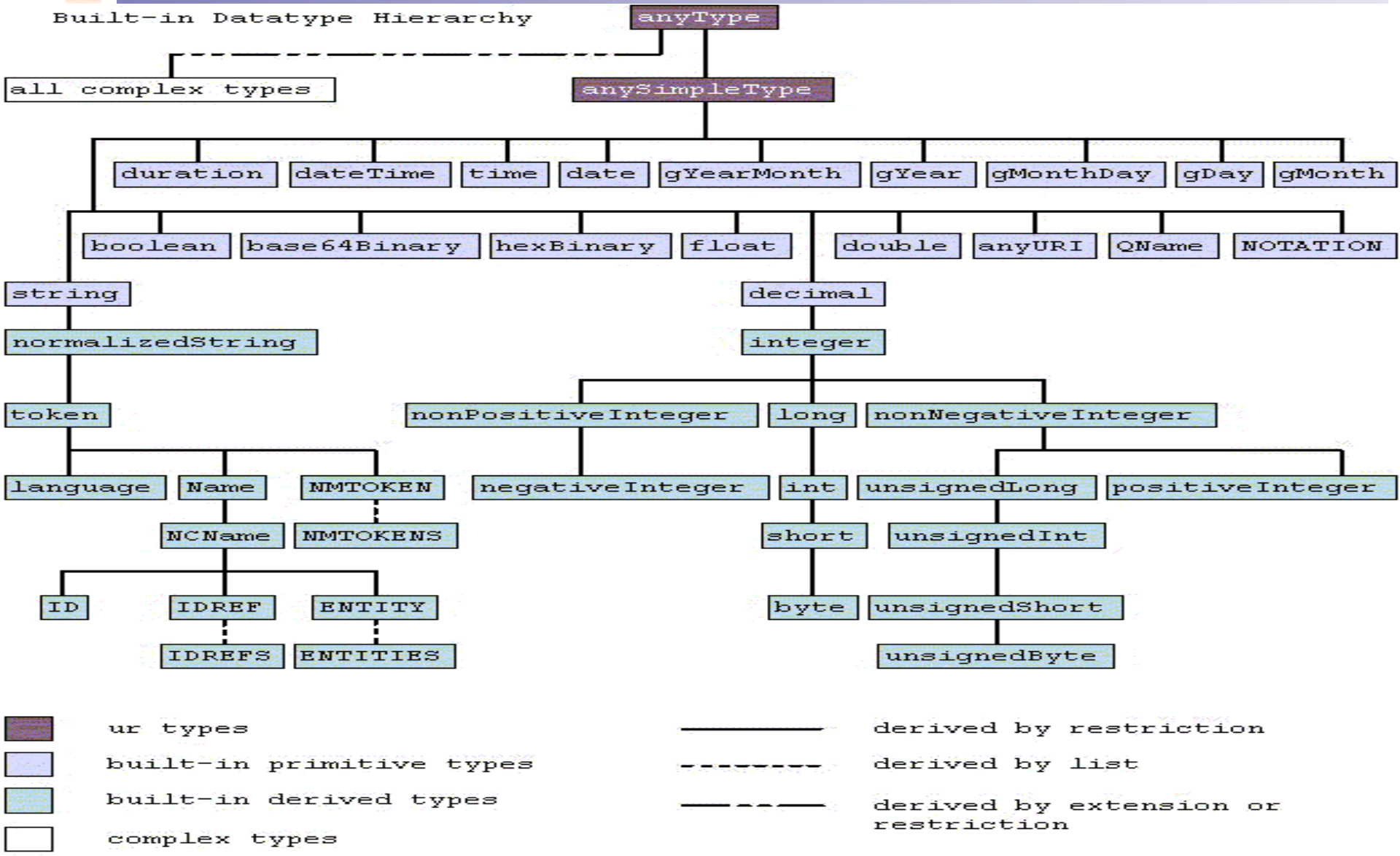
Primitive Datatypes	
string	"Hello World"
boolean	{true, false, 1, 0}
decimal	7.08
float	IEEE single-precision 32-bit,INF,-INF,NAN
double	IEEE double-precision 64-bit,INF,-INF,NAN
dateTime	format: *CCYY-MM-DD hh:mm:ss*
time	format: *hh:mm:ss.sss*
date	format: *CCYY-MM-DD*
gYearMonth	format: *CCYY-MM*
gYear	format: *CCYY*
gMonthDay	format: *--MM-DD*

INF = infinity
NAN = not-a-number

Built-in Simple Types

Derived Datatypes	
negativeInteger	negative infinity to -1
long	-92233… to 92233…
int	-21474… to 2147…
short	-32768 to 32767
byte	-127 to 128
nonNegativeInteger	0 to infinity
unsignedLong	0 to 18446…
unsignedInt	0 to 42949…
unsignedShort	0 to 65535
unsignedByte	0 to 255
positiveInteger	1 to infinity

Build in data type hierarchy



anyType Element

- The ***anyType:***

  represents an abstraction called the ur-type which is the base type from which all simple and complex types are derived.

- An ***anyType*** type does not constrain its content in any way.

Example: anyType Element

- **In schema:**

```
<xs:element name="x" type="xs:anyType"/>
```

- **In XML:**

```
<x/>

<x>Hello</x>

<x><y>welcome</y></x>
```

"User Defined" derived type

- **Using User defined types:**

  It can be simple types or complex types.

1. Simple Type:

- Do not have "*sub-element*" or "*attribute*".
- Can be derived from existing simple types
  - Built-in "string, integer, decimal --etc "
  - **Or/and** another user derived types.

1. derived Simple type

- **`xs:simpleType`**: schema element used to define a simple type:
  - It has attribute called **name** to define new type name.
  - It has many sub-elements to make new data type:
    - **`xs:restriction:`**
      - Defines constraints on a given data types.
    - **`xs:union:`**
      - Defines a collection of values from given simple data types.
    - **`xs:list:`**
      - Defines a list of values within single element.

        (Problem in String & spaces)

Example: String Facets (Enumeration)

- **How to create the following XML elements?!!**
  - `<weekday> Monday </weekday>` -----→Valid
  - `<weekday> Jun </weekday>` ----------→not Vaild

Example (cont'd)

In Schema file:

```
<xs:simpleType name="weekType">
   <xs:restriction base="xs:string">
      <xs:enumeration value="Sunday"/>
      <xs:enumeration value="Monday"/>
            ......
      <xs:enumeration value="Saturday"/>
   </xs:restriction>
</xs:simpleType>

<xs:element name="weekday" type="weekType"/>
```

Example: string facet (Pattern)

```
<xs:simpleType name="TelephoneNumber">
 <xs:restriction base="xs:string">
      <xs:pattern value="\d{3}-\d{4}"/>
 </xs:restriction>
</xs:simpleType>


<xs:element name="phone" type="TelephoneNumber"/>
```

1
2
3

1. This creates a new datatype called '**TelephoneNumber**'.
2. Elements of this type can hold string values.
3. The string must follow the pattern: **ddd-dddd.**

Regular Expression URLs:
http://www.siteexperts.com/tips/functions/ts23/page1.asp

Anonymous Type Definition

- **Anonymous Type Definition:**

  Needed when define local data types within an element.

```
<xs:element name="TelephoneNo">
    <xs:simpleType>
      <xs:restriction base="xs:string">
         <xs:pattern value="\d{3}-\d{4}"/>
      </xs:restriction>
   </xs:simpleType>
</xs:element>
```

Note:
No name attribute

Facets of the integer Datatype

- The integer data type optional facets:
  - totalDigits
  - pattern
  - enumeration
  - maxInclusive
  - maxExclusive
  - minInclusive
  - minExclusive

Example: integer facet

- **How to create the following XML elements?!!**
  - `<StudentBirthDay>20</StudentBirthDay>` valid
  - `<StudentBirthDay>32</StudentBirthDay>` non-valid

Example: integer facet (cont'd)

1. Creating Simple Type:

```
<xs:simpleType name="dayOfMonth">
      <xs:restriction base="xs:integer">
            <xs:minInclusive value="1"/>
            <xs:maxInclusive value="31"/>
      </xs:restriction>
</xs:simpleType>
```

2. Using The Created Simple type:

```
<xs:element name="StudentBirthDay" type="dayOfMonth"/>
```

Example of using "Union"

- **How to create the following XML elements?!!**
  - `<Jeans_size>valid-values</Jeans_size>`

  •where its valid values are:

  - Integer range "from 22 - To 42".
  - **or** a string "small | meduim | larg ".

Example of using "Union" (cont'd)

Schema representation:

```
<xs:simpleType name="sizebystring">
    <xs:restriction base="xs:string">
          <xs:enumeration value="small"/>
          <xs:enumeration value="medium"/>
          <xs:enumeration value="large"/>
    </xs:restriction>
 </xs:simpleType>

<xs:simpleType name="sizebyno">
    <xs:restriction base="xs:positiveInteger">
          <xs:minInclusive value="22"/>
          <xs:maxInclusive value="42"/>
    </xs:restriction>
</xs:simpleType>
```

Example of using "Union" (Cont'd)

```
<xs:element name="Jeans_size">
  <xs:simpleType>
    <xs:union memberTypes="sizebyno sizebystring"/>
  </xs:simpleType>
</xs:element>
```

Note:
No name attribute

- You can apply "Two facets" to a union type.
  pattern and enumeration,

Creating a simpleType from another simpleType

- we can create a simpleType that uses another simpleType as the base.

```
<xs:simpleType name= "CarPrice">
    <xs:restriction base="xs:integer">
        <xs:minInclusive value="20000"/>
        <xs:maxInclusive value="500000"/>
    </xs:restriction>
</xs:simpleType>
```

```
<xs:simpleType name= "FIATPrice">
      <xs:restriction base="CarPrice">
         <xs:minInclusive value="35000"/>
         <xs:maxInclusive value="70000"/>
      </xs:restriction>
</xs:simpleType>
```

Lab Exercise

Assignment

- Create XML-Schema to the following XML doc.

```
<WeekDay>valid-values</WeekDay>
```

where its valid-values are:

- Integer range “from 1 - To 7”.
- **or** a string “Saturday | Sunday | ….. | Thursday”.

"User Defined" derived type

- **Using User defined types:**

  It can be simple types or complex types.

2. Complex Type:

- Sub-Element declarations.
- *And/Or attribute* declarations.
- *And/Or element* references.
- It has attribute called Mixed.

2. Complex type

- **xs:complexType:** is a schema element used to define a Complex type element.

- It has many sub-elements to make new data type.

- **Example:**

  - xs:sequence.
  - xs:simpleContent.
  - xs:choice.
  - xs:group.
  - xs:attribute.
  - xs:complexContent.
  - xs:all.
  - xs:attributeGroup.

  And so on.

2. Complex type (cont'd)

- **xs:sequence**:
  - Requires that all element should occur with the same order mentioned.
- **xs:all**:
  - Requires that all element should occur irrespective of the order.
- **xs:choice**: [OR relationship]
  - Requires that only one of elements in the "choice clause" should occur.

Example

- **XML Nodes:**

```
<person>
   <firstname>Mohamed</firstname>
   <lastname>Ahmed</lastname>
</person>
```

- **Schema representation:**

```
<xs:element name="person">
   <xs:complexType>
      <xs:sequence> <!--(xs:all) or (xs:choice)-->
         <xs:element name="firstname" type="xs:string"/>
         <xs:element name="lastname" type="xs:string"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Elements Occurrences Examples

```
<xs:element name="Exams">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="test"  type="xs:string"
          minOccurs="1" maxOccurs="1"
or        minOccurs="2" maxOccurs="unbounded"
or        minOccurs="1" maxOccurs="1" fixed="Hi"
or        minOccurs="0" maxOccurs="1" default="Hi"/>
      <xs:element name="final" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Complex type Contains Mixed

- **XML Node:**

```
<Greetings>DearMr
    <name>RobertSmith</name>
</Greetings>
```

- **Schema Representation:**

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

Creating xs:attribute

- **xs:attribute**:

  is schema element used to define an attribute.

- **General form:**

```
<xs:attribute name="" optional-Attributes??/>
```

  – **Optional-Attributes:** include

  " type , default , fixed , use , … etc".

Creating xs:attribute (Cont'd)

Declare Default and Fixed Values for Attributes

- `<xs:attribute name="lang" type="xs:string" default="EN"/>`
  Valid

- `<xs:attribute name="lang" type="xs:string" fixed="EN"/>`
  Valid

- `<xs:attribute name="lang" fixed="Ar" default="EN"/>`
  Non Valid

Creating xs:attribute (Cont'd)

Legal values of use Attributes:

- ✓ All attributes are optional by default, but you can write

```
<xs:attribute name="lang" use="optional"/>
<xs:attribute name="lang" use="required"/>
<xs:attribute name="lang" use="prohibited"/>
```

- ✓ Prohibited can not appear.

Creating xs:attribute (Cont'd)

- XML node with empty content:
  - `<person age="24"/>`
- Schema representation of attributes only:

```
<xs:element name=" person" >
  <xs:complexType>
    <xs:attribute name="age" type="xs:integer"/>
  </xs:complexType>
</xs:element>
```

Creating xs:simpleContent

- **xs:simpleContent:**
  - It is used to indicate that the complex type contains only character data and attributes.
    - i.e. it can not contains sub-elements.

- **Example:**

```
<person age="24">ahmed</person>
```

Creating xs:simpleContent (Cont'd)

- **Schema representation:**

```
<xs:element name="person" >
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
      <!-- or < xs:restriction base="xs:string">-->
        <xs:attribute name="age" type="xs:integer"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element >
```

Contains Attributes and Elements Only

- **XML Nodes:**

```
<BillTo country="Eg">
   <name>Ahmed Mohamed</name>
   <street>241 AlAhram</street>
   <city>Giza</city>
   <state>Giza</state>
</BillTo>
```

Attributes and Elements Only (cont'd)

- **Schema representation:**

```
<xs:complexType name="EGAddress">
  <xs:sequence>
    <xs:element name="name"   type="xs:string"/>
    <xs:element name="street" type="xs:string"/>
    <xs:element name="city"   type="xs:string"/>
    <xs:element name="state"  type="xs:string"/>
  </xs:sequence>
  <xs:attribute name="country" type="xs:NMTOKEN"
   fixed="EG"/>
</xs:complexType>

<xs:element name="BillTo" type="EGAddress"/>
```

Derived Complex type "xs:complexContent"

- We can do a form of subclassing complexType definitions. We call this "derived types":
  - Derive by **extension**: extends the parent complexType with more element.
  - Derive by **restriction**: creates a type which is a subset of the base type.
  - **Derivation** by extension **or** restriction **is done using :**
    - **xs:complexContent**

Derived by extension

- **Parent Element:**

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

Derived by extension (Cont'd)

- **Child Element:**

```
<xs:complexType name="BookPublication">
  <xs:complexContent>
    <xs:extension base="Publication" >
      <xs:sequence>
        <xs:element name="ISBN‟ type="xs:string"/>
        <xs:element name="Publisher "
                    type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Derived by Restriction

- **Parent Element:**

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

Derived by Restriction (Cont'd)

```
<xs:complexType name= "SingleAuthorPublication">
  <xs:complexContent>
    <xs:restriction base="Publication">
      <xs:sequence>
        <xs:element name="Title"  type="xs:string"/>
        <xs:element name="Author" type="xs:string"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

- **Child Element:**
  - You can change parent element type.
  - You can not add new elements to child.

Prohibiting Derivations

- Publication cannot be extended nor restricted:

```
<xs:complexType name = "Publication"
                final= "#all" >
```

- Publication cannot be restricted:

```
<xs:complexType name = "Publication"
                final= "restriction" >
```

- Publication cannot be extended:

```
<xs:complexType name = "Publication"
                final= "extension" >
```

Element Repetition

```
<xs:element name="Second">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="First" type="xs:string"/>
      <xs:element name="Third" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="Fourth">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="First" type="xs:string"/>
      <xs:element name="Fifth" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Global Reference

```
<xs:element name="First" type="xs:string" />
<!-------------------------------------------->
<xs:element name="Second">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="First"/>
      <xs:element name="Third" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<!-------------------------------------------->
<xs:element name="Fourth">
  <xs:complexType>
    <xs:sequence>
        <xs:element ref="First"/>
        <xs:element name="Fifth" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

The Group Element

```
<xs:group name="custGroup">
  <xs:sequence>
    <xs:element name="customer" type="xs:string"/>
    <xs:element name="orderdetails"
                type="xs:string"/>
    <xs:element name="billto" type="xs:string"/>
    <xs:element name="shipto" type="xs:string"/>
  </xs:sequence>
</xs:group>
```

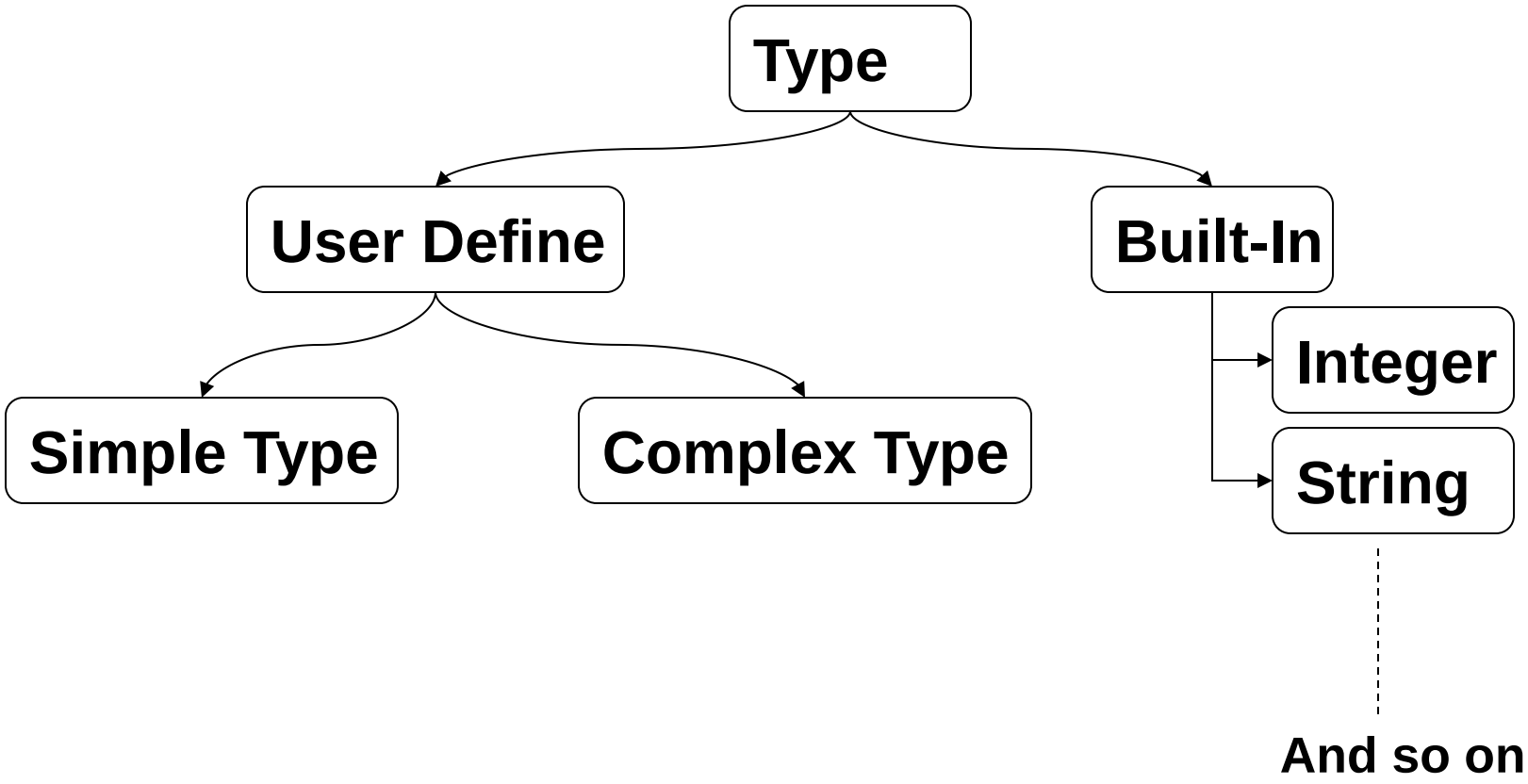
```
<xs:element name="order">
  <xs:complexType>
    <xs:group ref="custGroup"/>
    <xs:attribute name="status" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

Attribute Groups

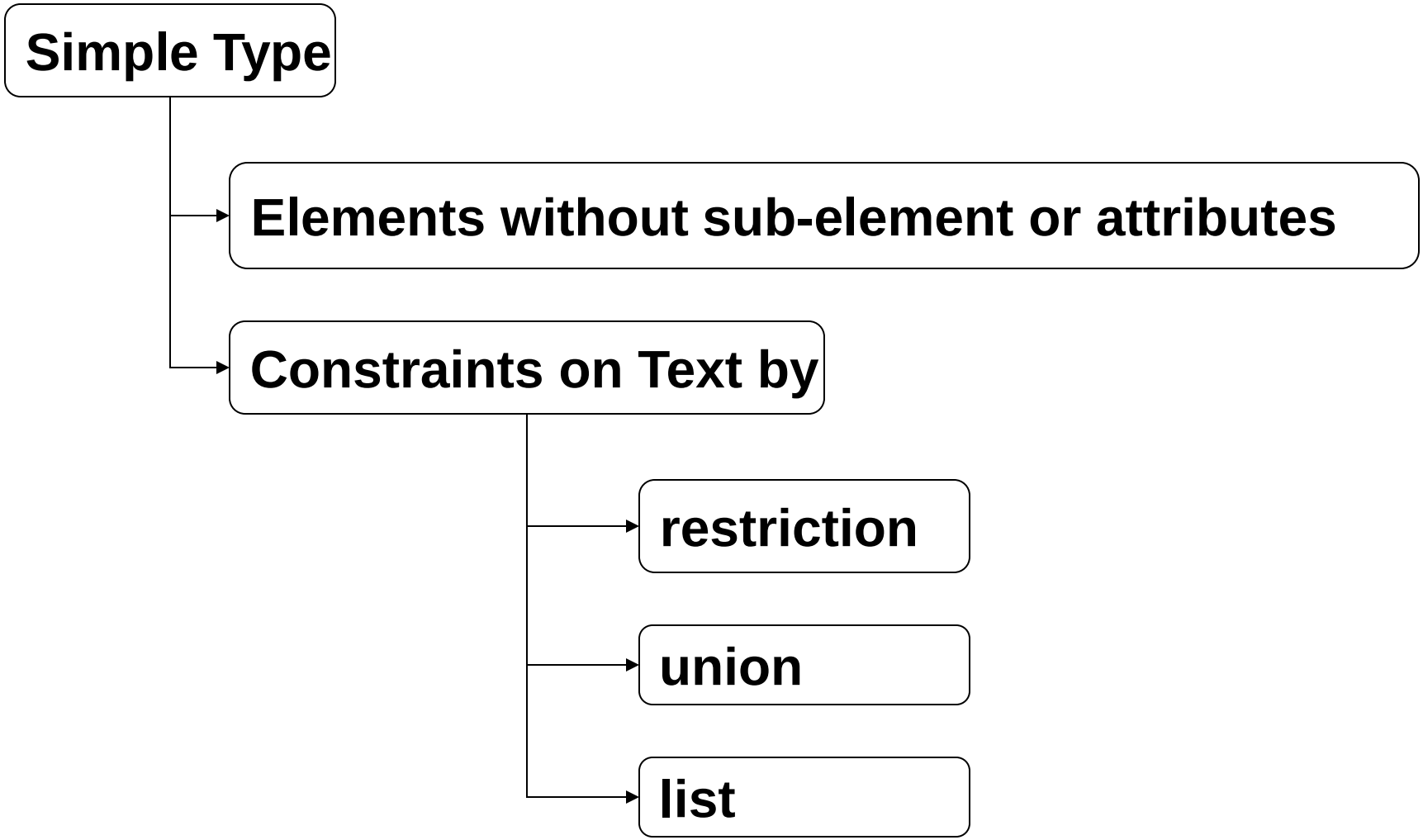
```
<xs:attributeGroup name="personattrgroup">
  <xs:attribute name="firstname" type="xs:string"/>
  <xs:attribute name="lastname" type="xs:string"/>
  <xs:attribute name="birthday" type="xs:date"/>
</xs:attributeGroup>
<!-------------------------------------------->
<xs:element name="person">
  <xs:complexType>
    <xs:attributeGroup ref="personattrgroup"/>
  </xs:complexType>
</xs:element>
```

Schema Type Revision

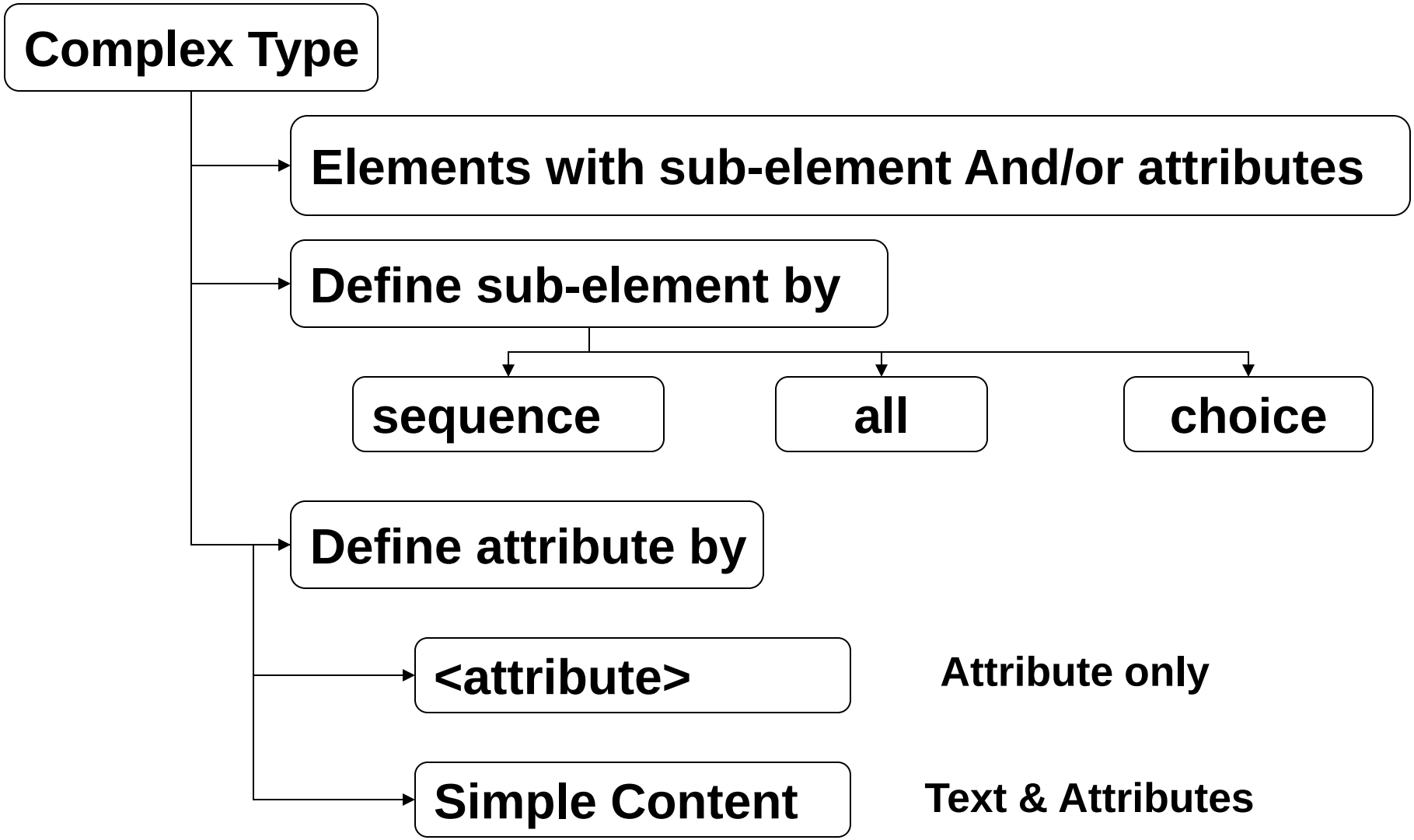
```
<xs:element name="Price" type="xs:decimal"/>
```



Schema Type Revision (Cont'd)



Schema Type Revision (Cont'd)



Lab Exercise

Assignment 1

- Design a schema of the configuration file for a library that you made it.
- Note:
  - In schema:
    - Use Simple type
    - Use complex type (Simple content, Complex content)
    - Group (element, attribute)