XML Schema

What is XML Schema?

- The origin of schema
  - XML Schema documents are used to define and validate the content and structure of XML data.
  - XML Schema was originally proposed by Microsoft, but became an official W3C recommendation in May 2001

DTD versus Schema

Limitations of DTD

- No constraints on character data
- Not using XML syntax
- No support for namespace
- Very limited for reusability and extensibility

Advantages of Schema

- Syntax in XML Style
- Supporting Namespace and import/include
- More data types
- Able to create complex data type by inheritance
- Inheritance by extension or restriction
- More …

An XML Instance Document Example

```
<book isbn="0836217462">
    <title> Being a Dog Is a Full-Time Job</title>
    <author>Charles M. Schulz</author>
    <qualification>    extroverted beagle  </qualification>
</book>
```

The Example's Schema

```
<?xml version="1.0" encoding="utf-8"?>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="book">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element name="author" type="xs:string"/>
            <xs:element name="qualification" type="xs:string"/>
          </xs:sequence>
<xs:attribute name="isbn" type="xs:integer"/>
        </xs:complexType>
      </xs:element>
</xs:schema>
```

book.xsd

DTD v.s. XML Schemas

DTD:

```
<!ELEMENT paper (title,author*,year, (journal|conference))>
```

XML Schema:

```
<xs:element name="paper">
<xs:complexType >
    <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="author" minOccurs="0"/>
        <xs:element name="year"/>
        <xs: choice> < xs:element name="journal"/>
                        <xs:element name="conference"/>
        </xs:choice>
    </xs:sequence>
</xs:complexType >
</xs:element>
```

Example

A valid XML Document:

```
<paper>
  <title> The Essence of XML </title>
  <author> Simeon</author>
  <author> Wadler</author>
  <year>2003</year>
  <conference> POPL</conference>
</paper>
```

General structure

- To be able to mix tags from different grammars, we use namespaces to avoid possible ambiguities.
- The XML schema namespace is identified by the URI: http://www.w3.org/2001/XMLSchema.

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

  <!-- Déclarations d'éléments, d'attributs et définitions de types
-->
  ...
</xsd:schema>
```

Element declarations

- To declare an element you must use the <xsd:element> tag
- The simplest form is as follows:

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

- The value of the name attribute = name that the element will have in the doc xml
- The value of the type attribute = the type of the element.

The element type can be:

- a simple predefined type (e.g. xsd:string, xsd:integer, etc.)
- a type defined by ourselves elsewhere in the diagram

Element declarations: without explicit reference to a type

It is not required to explicitly reference a type via the type attribute in the xsd:element element.

- In this case, the element type must be described in the xsd:element tag.
- The syntax is then one of the following two depending on whether the type is simple or complex.

```
<xsd:element name="item">
  <xsd:simpleType>
  ...
  </xsd:simpleType>
</xsd:element>
```

```
<xsd:element name="item">
  <xsd:complexType>
  ...
  </xsd:complexType>
</xsd:element>
```

Element declarations: default value (resp. fixed)

- When the type of the declared element is predefined or simple (<xsd:simpleType>), it can be given a default value via the default attribute.

```
<xsd:element name="integer" type="xsd:integer" default="7‘’/>
```

- We can also force an element to have an attribute that always has the same value via the fixed attribute.

```
<xsd:element name="number" type="xsd:integer" fixed="7"/>
```

Type definitions

The types can be classified into 2 categories:

- Simple types.
  - They only define textual content.
  - They can be used to type attributes and elements
  - They are subdivided into 2 subcategories:
    - the predefined ones.
    - The non-predefined.
- Complex types

  Complex types with simple content.

  Complex types with complex content.

Local v.s. Global Types

- Local type:

```
<xs:element name="person">
        [define locally the person's type]
</xs:element>
```

- Global type:

```
<xs:element name="person" type="ttt"/>

<xs:complexType name="ttt">
        [define here the type ttt]
</xs:complexType>
```

Global types: can be reused in other elements

Local v.s. Global Elements

- Local element:

```
<xs:complexType name="ttt">
    <xs:sequence>
        <xs:element name="address" type="..."/>...
    </xs:sequence>
</xs:complexType>
```

- Global element:

```
<xs:element name="address" type="..."/>

<xs:complexType name="ttt">
    <xs:sequence>
        <xs:element ref="address"/>  ...
    </xs:sequence>
</xs:complexType>
```

Elements v.s. Types

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name"
                  type="xs:string"/>
      <xs:element name="address"
                  type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="person"
                  type="ttt">
<xs:complexType name="ttt">
  <xs:sequence>
    <xs:element name="name"
                  type="xs:string"/>
    <xs:element name="address"
                  type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Both say the same thing; in DTD:

<!ELEMENT person (name,address)>

Simple types

- string
  - Confirm this is electric
- normalizedString
  - Confirm this is electric
- token
  - Confirm this is electric
- byte
  - -1, 126
- unsignedByte
  - 0, 126
- base64Binary
  - GpM7
- hexBinary
  - 0FB7
- integer
  - -126789, -1, 0, 1, 126789
- positiveInteger
  - 1, 126789
- negativeInteger
  - -126789, -1
- nonNegativeInteger
  - 0, 1, 126789
- nonPositiveInteger
  - -126789, -1, 0
- int
  - -1, 126789675
- unsignedInt
  - 0, 1267896754

Simple types

- long
  - -1, 12678967543233
- unsignedLong
  - 0, 12678967543233
- short
  - -1, 12678
- unsignedShort
  - 0, 12678
- decimal
  - -1.23, 0, 123.4, 1000.00
- float
  - -INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
- double
  - -INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
- boolean
  - true, false 1, 0
- time
  - 13:20:00.000, 13:20:00.000-05:00
- dateTime
  - 1999-05-31T13:20:00.000-05:00
- duration
  - P1Y2M3DT10H30M12.3S
- date
  - 1999-05-31
- gMonth
  - --05--
- gYear
  - 1999

Simple types

- gYearMonth
  - 1999-02
- gDay
  - ---31
- gMonthDay
  - --05-31
- Name
  - shipTo
- QName
  - po:USAddress
- NCName
  - USAddress
- anyURI
  - http://www.example.com/,
  - http://www.example.com/doc.html#ID5
- language
  - en-GB, en-US, fr
- ID
  - "A212"
- IDREF
  - "A212"
- IDREFS
  - "A212" "B213"
- ENTITY
- ENTITIES
- NOTATION
- NMTOKEN, NMTOKENS
  - US
  - Brésil Canada Mexique

Simple non-predefined types

- Example of a simple type declaration:

```
<xsd:simpleType name="major">
    <xsd:restriction base="xsd:unsignedByte">
        <xsd:minInclusive value="18"/>
    </xsd:restriction>
</xsd:simpleType>
```

Local Names

name has different meanings in **person** and in **product**

```
<xs:element name="person">
 <xs:complexType>
      . . . . .
    <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:element>
     . . . .
 </xs:complexType>
</xs:element>


<xs:element name="product">
 <xs:complexType>
      . . . . .
    <xs:element name="name"   type="xs:string"/>

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

Complex types

 Complex types are divided into 2 categories.

– Complex types with simple content: they are similar to simple non-predefined types but with an additional addition: the definition of attributes via the xsd:attribute element. At least one attribute must be described.

– Complex types with complex content: they define contents made up of other elements. The content is either mixed (i.e., elements + text), or homogeneous (i.e., only a possibly empty set of elements). They can also include attribute descriptions but this is not obligatory.

Complex types

- Complex types can only be used to describe the type of elements. They cannot be used to type attributes.
- A complex type can be constructed in 2 ways:
  - explicitly
  - implicitly by deriving it from another simple or complex type (cf. extension or restriction operations).
- In all cases, to define a complex type you must use the xsd:complexType element

Complex types

- To construct a complex type explicitly, we use the following operators:
  - xsd:sequence
  - xsd:choice
  - xsd:all

```
<xs:complexType name="PurchaseOrderType">
 <xs:sequence>  <xs:element name="shipTo" type="USAddress"/>
              <xs:element name="billTo" type="USAddress"/>
              <xs:element ref="comment" minOccurs="0"/>
              <xs:element name="items"  type="Items"/>
 </xs: sequence >
 <xs:attribute name="orderDate" type="xs:date"/>
</xs:complexType>
```

“Mixed” Content, “Any” Type

```
<xs:complexType mixed="true">
  . . . .
```

- Better than in DTDs: can still enforce the type, but now may have text between any elements

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

- Means anything is permitted there

“All” Group

```
<xs:complexType name="PurchaseOrderType">
  <xs:all>  <xs:element name="shipTo" type="USAddress"/>
            <xs:element name="billTo" type="USAddress"/>
            <xs:element ref="comment" minOccurs="0"/>
            <xs:element name="items"  type="Items"/>
  </xs:all>
  <xs:attribute name="orderDate" type="xs:date"/>
</xs:complexType>
```

- A restricted form of & in SGML
- Restrictions:
  - Only at top level
  - Has only elements
  - Each element occurs at most once
- E.g. “comment” occurs 0 or 1 times

Derived Types by Extensions

```
<complexType name="Address">
    <sequence>  <element name="street" type="string"/>
                <element name="city"   type="string"/>
    </sequence>
</complexType>

<complexType name="USAddress">
  <complexContent>
     <extension base="ipo:Address">
        <sequence>  <element name="state" type="ipo:USState"/>
                    <element name="zip"   type="positiveInteger"/>
        </sequence>
     </extension>
  </complexContent>
</complexType>
```

Derived Types by Restrictions

```
<complexContent>
    <restriction base="ipo:Items“>
     … [rewrite the entire content, with restrictions]...
    </restriction>
 </complexContent>
```

- (*): may restrict cardinalities, e.g. (0,infty) to (1,1); may restrict choices; other restrictions…

Corresponds to set inclusion

Regular Expressions

Recall the element-type-element alternation:

```
<xs:complexType name="....">
        [regular expression on elements]
</xs:complexType>
```

Regular expressions:

- <xs:sequence> A B C </...> = A B C
- <xs:choice> A B C </...> = A | B | C
- <xs:group> A B C </...> = (A B C)
- <xs:... minOccurs="0" maxOccurs="unbounded"> ..</...> = (...)*
- <xs:... minOccurs="0" maxOccurs="1"> ..</...> = (...)?

The patterns

- Constraints on predefined simple type
- Use of regular expressions
- Example

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

Facets of Simple Types

- Facets = additional properties restricting a simple type
- 15 facets defined by XML Schema

Examples

- length
- minLength
- maxLength
- pattern
- enumeration
- whiteSpace

- maxInclusive
- maxExclusive
- minInclusive
- minExclusive
- totalDigits
- fractionDigits

Types reuse

- Simple type extension :

```
<xs:simpleType name="num5">
    <xs:restriction base="xs:string">
        <xs:pattern value="\d{5}"/>
    </xs:restriction>
</xs:simpleType>
```

- Complexe type (sequence):

```
<xs:element name="livre">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Titre" type="xs:string"/>
            <xs:element name="Auteur" type="xs:string"/>
            <xs:element name="ISBN" type="num5"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Attributes

- The definition of attributes associated with an element is done in the tag *<attribute>.*
  - *name* :
  - *type* : attribute type, can only be a simple type
  - *use* : specify whether the attribute is required, optional or prohibited.

  Possible values:
    - *required*
    - *optional*
    - *prohibited*
  - *fixed* : do not change
  - *default* : default value.

Attributes

- Attributes definition can be done in two different ways.
  - Define the attribute and use it in the type definition.
  - Define the attribute directly inside the type,

```
<xsd:attribute name="sexe">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="F"/>
            <xsd:enumeration value="M"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:attribute>

<xsd:element name="employee">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="nom" type="xsd:string" />
            <xsd:element name="contact">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="empAdresse" type="tns:AdresseOnlyAttributes" />
                        <xsd:element name="phone" type="xsd:string" />
                        <xsd:element name="email" type="tns:mailType" />
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="dateMAJ" type="xsd:dateTime" />
        <xsd:attribute ref="tns:sexe"/>
    </xsd:complexType>
</xsd:element>
```

Elements grouping

```
<xs:group name="TitreAuteurISBN">
    <xs:sequence>
        <xs:element name="Titre" type="xs:string"/>
        <xs:element name="Auteur" type="xs:string"/>
        <xs:element name="ISBN" type="num5"/>
    </xs:sequence>
</xs:group>
<xs:element name="livre">
    <xs:complexType>
        <xs:sequence>
            <xs:group ref="TitreAuteurISBN"/>
            <xs:element ref="traduction"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

XML Schema Reference

- Reference without namespace

```
<Livre xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="Livre.xsd">
  <Titre>Les réseaux</Titre>
    ...
```

- Reference with namespace

```
<bib:Livre xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:bib="http://www.cpe.fr/ns/bib"
           xsi:schemaLocation="http://www.cpe.fr/ns/bib Livre.xsd">
  <bib:Titre>Les réseaux</bib:Titre>
```