XML Schema languages

HAP/LAP. Corpus Linguistics.

Conceptual Schemas

- XML document class (application) vs. document instances.
- To define document types: schemas.
- Schema language: language to describe/specify schemas.
- Schema processing:
  - validation: test whether an XML document complies with the schema
  - normalization: if the document is valid, a normalized version is obtained; after normalization: *document instance*

Schema languages

- Document Type Definition (DTD)
  - schema language defined in XML 1.0, it comes from SGML
  - most widely used (at the moment?)
  - narrative-oriented
- W3C XML Schema
  - data-oriented
  - richer and more powerful
  - written in XML
- Relax NG
  - simpler to learn
  - good support for unordered content
  - written in XML, but it has also a compact form

Document Type Definition (DTD)

- DTD specifies:
  - which element can appear (or must appear) in the document
  - attributes of elements
  - entities
  - context on which any item may/has to appear

Example of a simple DTD

```
<!ELEMENT person (nameSurname, occupation*) >
<!ELEMENT nameSurname (name, surname) >
<!ELEMENT name (#PCDATA) >
<!ELEMENT surname (#PCDATA) >
<!ELEMENT occupation (#PCDATA) >
```

- Usually, DTDs are a separate files (`.dtd` extension)
  - many XML documents point to the same DTD
- A DTD is a list of element declarations.
- Order is not important. There maybe forward and backward references, as well as circular definitions.

Validity of documents

Valid (against DTD)

```
<person>
  <nameSurname>
    <name>Pier Paolo</name>
    <surname>Pasolini</surname>
  </nameSurname>
  <occupation>writer</occupation>
</person>
```

Invalid

```
<person>
  <nameSurname>
    <name>Pier</name>
    <name>Paolo</name>
    <surname>Pasolini</surname>
  </nameSurname>
  <occupation>
    writer
  </occupation>
</person>
```

```
<person>
  <nameSurname>
    <surname>Pasolini</surname>
  </nameSurname>
  <occupation>
    writer
  </occupation>
</person>
```

Referencing the DTD

- Document Type Declaration
  - beginning of XML document

```
<!DOCTYPE person SYSTEM http://www.example.com/dtds/person.dtd >
```

- In the declaration:
  - which is the *root element* (`person`)
  - URI of the DTD file (may be a local file)

Referencing the DTD

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE person SYSTEM "person.dtd" >

<person>
  <nameSurname>
    <name>Pier Paolo</name>
    <surname>Pasolini</surname>
  </nameSurname>
  <occupation>writer</occupation>
</person>
```

- The first two lines are called the *prolog* section.

DTD inside

```
<!DOCTYPE person SYSTEM "person.dtd" [
  <!ELEMENT person (nameSurname, occupation*) >
  <!ELEMENT nameSurname (name, surname) >
  <!ELEMENT name (#PCDATA) >
  <!ELEMENT surname (#PCDATA) >
  <!ELEMENT occupation (#PCDATA) >
]>
```

- The whole DTD can be put inside the prolog section.

Element declaration

```
<!ELEMENT element-name element-content >
```

- Element name: any XML name.
- Element content:
  - `EMPTY`: empty element
  - `ANY`: any content
  - `#PCDATA`: textual content
  - Elements children:

    ```
    <!ELEMENT fax (telNumber) >
    ```

  - Sequences:

    ```
    <!ELEMENT nameSurname (name, surname) >
    ```

Element declaration

- Element content (cont.):
  - options:

```
<!ELEMENT publication (paper | journal) >
```

  - mixed content:

```
<!ELEMENT text (#PCDATA | comment) >
```

  - cardinality:
    - 1, if not specified
    - `?`: 0 or 1
    - `*`: 0 or more
    - `+`: 1 or more

```
<!ELEMENT telNumbers (telNumber)+ >
```

Attribute declaration

```
<!ATTLIST element-name attribute-name
        attrib-type default-value >
```

- Define the attributes of elements.
- Define many attributes inside the `ATTLIST` declaration.

```
<!ATTLIST image
        source CDATA #REQUIRED
        width CDATA #REQUIRED
        heigth CDATA #REQUIRED
        comment CDATA #IMPLIED>
```

Attribute types

- `CDATA`: general type, any string
- `NMTOKEN`: similar to XML names (but can start with a digit), no whitespace allowed
- `NMTOKENS`: list of tokens separated by spaces:

```
<cl_class dates="07-07-2014 08-08-2015 09-09-2016"/>
```

- Enumerations: exhaustive list of possible values.

```
<!ATTLIST week day (monday | tuesday | wednesday |
thursday | friday | saturday | sunday ) #REQUIRED >
```

Attribute types

- `ID`: identifier of element
  - the value is an XML name (can NOT start with digit)
  - no two elements with same id value in the same document
  - elements can have at most one attribute of type `ID`

```
<!ATTLIST person idcard ID #REQUIRED >
```

- `IDREF`: references an ID of another element
  - similar to foreign key
  - used to reference other elements
- `IDREFS`: list of `IDREFS` separated by spaces

```
<person idcard="_123456">...</person>
...
<project code="p10">....</project>
...
<student perId="_123456" projCode="p10 p23"/>
```

Attribute types

- These types are rare:
  - `ENTITY`: the value is an entity defined in the same DTD
  - `ENTITIES`: a space separated list of entities
  - `NOTATION`: the value is a notation defined in the same DTD

Attribute default values

- Four options:
  - `#IMPLIED`: the attribute is optional, no default value
  - `#REQUIRED`: the attribute is required, no default value
  - `#FIXED`: the attribute value is a fixed literal (constant)

```
<!ATTLIST biography
   xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink">
```

  - Literal: the default value, in quotes

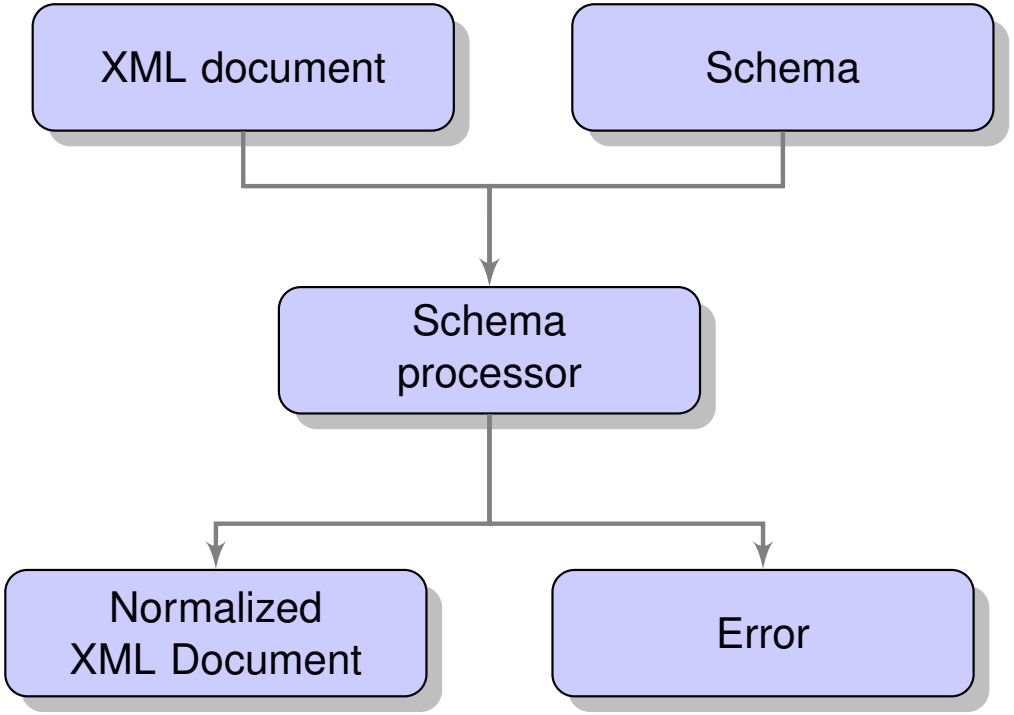
Entity declaration

- There are five pre-defined entities
- More can be defined:

```
<!ENTITY entity-name entity-value>
```

- Inner entity : `<!ENTITY WWW "World Wide Web">`
- External entity : `<!ENTITY footer SYSTEM "footer.xml">`
  - uses an URI to reference a document, which is inserted whenever the entity is used
- Using references in documents: `&entity_name;`
  - `&WWW; &footer; ...`
  - define once, use as many times as needed

Validation


XML document
Schema
Schema processor
Normalized XML Document
Error

Valid vs. well formed

- XML document is well formed:
  - follows XML syntax
  - can be processed
- XML document is valid:
  - against a particular schema
  - content follows schema specification

Standarized DTDs

- There are many standarized DTDs for different purposes
  - no centralized repository, though
- Use them whenever possible
  - but be aware of license issues
- Some examples:
  - TEI (*Text Encoding Initiative*): `http://www.tei-c.org`
  - DocBook `http://http://docbook.org`
  - RSS
  - ...

DTD problems

1. Does not use XML syntax.
2. Can not put restrictions on textual content.
3. The model for attributes/values is too simple.
4. No namespaces.
5. Weak modularity and reusability.
6. No inheritance.
7. No way to document things.
8. Text and attribute declarations are content independent.
9. ID attributes are too simple.
10. No default for elements (only attributes).
11. Simple cardinality (zero, one, many).

DTD: in summary

- Enough for narrative-oriented document
- Weak for data-oriented documents

XML namespaces

- Avoid conflicts when trying to mix different XML applications (schemas)
- Element and attribute names belong to namespaces
- Can mix elements with same name but different namespace

XML namespaces

- Elements and attributes may have a namespace URI
- Namespace URI: namespace *prefix* + name

XML namespaces

- Namespaces are specified with the `xmlns` attribute

```
<svg xmlns="http://www.w3.org/TR/SVG/"
    width="12cm" height="10cm">
  <ellipse rx="110" ry="130" />
  <rect x="4cm" y="1cm" width="3cm" height="6cm" />
</svg>
```

- We assign a namespace to the `svg` element
  - all children inherit the namespace

XML namespaces: prefix

- Element and attributes can start with a prefix: namespace
- Use `xmlns:` attribute to declare prefix and link to the URI

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="pertsona">
    <p> Pertsona bat </p>
  </xsl:template>
</xsl:stylesheet>
```

- This example shows the definition and use of the `xsl` prefix (XSLT)

XML namespaces: example

```
<?xml version="1.0"?>
<!-- namespaces previously declared -->
<bk:book xmlns:bk='urn:loc.gov:books'
         xmlns:isbn='urn:ISBN:0-395-36341-6'>
 <bk:title>XML in a nutshell</bk:title>
   <isbn:number>1568491379</isbn:number>
</bk:book>
```

XML Schema

Design principles and main characteristics:

1. written in XML
   - there is an XML Schema for XML Schema!
2. more expression power than DTDs:
   - mechanisms to put restrictions on structure and data
   - namespaces
   - pre-defined data types
   - object-oriented type system
   - modularity
   - restrictions on cardinality
   - more powerful than `ID`/`IDREF` attributes
   - regular expressions to specify restrictions on content

XML Schema: example

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
  <xs:element name="name" type="xs:string"></xs:element>
</xs:schema>
```

- Conforming document:

```
<?xml version="1.0" encoding="UTF-8"?>
<name>
  John Doe
</name>
```

Element definition

- Use `<xs:element>` element.
- Can have many attributes. Among others:
  - `name`: name of the element.
  - `type`: element type. Can be simple or complex type.
  - `id`: ID type.
  - `maxOccurs`: how many times it may occur at most. Default is 1. Use `unbounded` to put no limits.
  - `minOccurs`: how many times it must occur at least. Default is 1. Use `unbounded` to put no limits.

XML Schema: another example

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
  <xs:element name="shipOrder" type="shipType"/>
  <xs:complexType name="shipType">
    <xs:sequence>
      <xs:element name="address" type="xs:normalizedString"/>
      <xs:element name="comments" minOccurs="0"/>
      <xs:element name="quantity" type="xs:positiveInteger"/>
      <xs:element name="price" type="xs:decimal"/>
    </xs:sequence>
    <xs:attribute name="date" type="xs:date"/>
  </xs:complexType>
</xs:schema>
```

Simple types

- Basic data types.
- Used on leaf elements (just text, no children).
- Can not have nested elements.
  - `xs:string`
  - `xs:integer`
  - `xs:boolean` (`true` or `false`)
  - `xs:anyURI`
  - `xs:dateTime` (combination date plus time. For inst: `2015-07-20T16:15:00-05:00`)
  - `xs:duration` (time span: year, month, day, week, ...)
  - `ID, IDREFS, ENTITY, ENTITIES, NOTATION, NMTOKEN, NMTOKENS` (XML 1.0)
  - `xs:language` (attribute `xml:lang` of XML 1.0)
  - `xs:normalizedString` (spaces instead of newlines and tabs)
  - `xs:token` like above, but without spaces.

Complex types

- Complex types: elements containing nested elements .
- Use the element `<xs:complexType>`
  - the type can be named, and used more than once.
  - If `<xs:complexType>` is used within an element, the type is anonymous (the `type` attribute is not used)
- The type specifies the structure of the content.

Content

- Empty: empty element, no content. For example:

```
<xs:element name="telephone" minOccurs="0">
  <xs:complexType>
    <xs:attribute name="number" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

- Any content: `<xs:any>`
- Complex content: only (sub-)elements.
- Simple context: text, no children.

Content (II)

- Mixed content: when textual content and sub-elements are mixed. The `mixed` attibute has a `true` value.
- Ordering of the sub-elements:
  - `<xs:sequence>`: fixed order.
  - `<xs:choice>`: one element contained in the `choice` declaration.
  - `<xs:all>`: all elements must appear once.

Mixed content: example

```
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="opening">
        <xs:complexType mixed="true">
          <xs:choice>
            <xs:element name="hello" fixed="Hello"/>
            <xs:element name="bye"/>
          </xs:choice>
        </xs:complexType>
      </xs:element>
      <xs:element name="body">
        <xs:complexType mixed="true">
          <xs:all>
            <xs:element name="when" type="xs:date"/>
            <xs:element name="subject" type="xs:string"/>
          </xs:all>
        </xs:complexType>
      </xs:element>
      <xs:element name="closing" fixed="Farewell"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Element declaration and use

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

<xs:complexType name="card1">
  <xs:sequence maxOccurs="unbounded">
    <xs:element ref="name"/>
    <xs:element ref="address" minOccurs="1"/>
    <xs:element ref="telephone" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="card2">
  <xs:sequence>
    <xs:element ref="name"/>
    <xs:element ref="telephone" minOccurs="1"/>
  </xs:sequence>
</xs:complexType>
```

Attributes

- Attribute declaration: `<xs:attribute>`
- Most used attributes for the `<xs:attribute>` element:
  - `type`: the attribute type (simple or built-in).
  - `default`: default value of attribute.
  - `fixed`: fixed value for attribute.
  - `id`: the attribute is and ID attribute.
  - `use`: how the attribute is used
    - `optional` (default)
    - `required`
    - `prohibited`

```
<xs:attribute name="age" type="xs:positiveInteger" use="required"/>
```

Creating new simple types

- To create new simple types, use the `<xs:simpleType>` element, and specify:
  - `<xs:restriction>` (also called *facets*): define the acceptable values for the new type.
  - `<xs:list>`: whitespace separated sequence of simple types.
  - `<xs:union>`: union of simple types

New simple types example

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

<xs:element name="weekEnd">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="saturday"/>
      <xs:enumeration value="sunday"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

New simple types example

```
<xs:element name="link">
  <xs:complexType>
    <xs:attribute name="confidence" type="zeroone"/>
  </xs:complexType>
</xs:element>

<xs:simpleType name="zeroone">
  <xs:restriction base="xs:float">
    <xs:minInclusive value="0"/>
    <xs:mAXInclusive value="1"/>
  </xs:restriction>
</xs:simpleType>
```

New simple types example

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

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

<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>
```

Facets

- As said before, facets are new types created by restricting simple types.
- Facet types:
  - `xs:length` (or `xs:minLength`, `xs:maxLength`): length of string (minimum, maximum).
  - `xs:pattern`: regular expression a string must match.
  - `xs:enumeration`
  - `xs:whiteSpace`: guidelines for dealing with white spaces. Three values: `preserve`, `replace` (replacer tabs and newlines with white spaces), `collapse` (same as `replace`, but replacing with just one white space).
  - `xs:maxInclusive` and `xs:maxExclusive` for numeric values.
  - `xs:minInclusive` and `xs:minExclusive`.
  - `xs:totalDigits`: number of digits of a numeric value.
  - `xs:fractionDigits`: number of fractional digits.

Facets example

```
<xs:simpleType name="order-num">
  <xs:restriction base="xs:string">
    <xs:pattern value="\d\d-\d-\d\d\d\d\d"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="name">
  <xs:restriction base="xs:string">
    <xs:maxLength value="50"/>
  </xs:restriction>
</xs:simpleType>
```