Chapter 4

XML Schema Definition Language (XSDL)

XML Schema

- XML Schema is a W3C Recommendation
  - XML Schema Part 0: Primer
  - XML Schema Part 1: Structures
  - XML Schema Part 2: Datatypes
- describes permissible contents of XML documents
- uses XML syntax
- sometimes referred to as *XSDL: XML Schema Definition Language*
- addresses a number of limitations of DTDs

Simple example

- file `greeting.xml` contains:

```
<?xml version="1.0"?>
<greet>Hello World!</greet>
```

- file `greeting.xsd` contains:

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

- `xsd` is prefix for the namespace for the "schema of schemas"
- declares element with name `greet` to be of built-in type `string`

DTDs vs. schemas

DTD	Schema
`<!ELEMENT>` declaration	`xsd:element` element
`<!ATTLIST>` declaration	`xsd:attribute` element
`<!ENTITY>` declaration	n/a
`#PCDATA` content	`xsd:string` type
n/a	other data types

Linking a schema to a document

- `xsi:noNamespaceSchemaLocation` attribute on root element
- this says no target namespace is declared in the schema
- `xsi` prefix is mapped to the URI:
  `http://www.w3.org/2001/XMLSchema-instance`
- this namespace defines global attributes that relate to schemas and can occur in instance documents
- for example:

```
<?xml version="1.0"?>
<greet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="greeting.xsd">
  Hello World!
</greet>
```

Validating a document

- W3C provides an XML Schema Validator (XSV)
- URL is `http://www.w3.org/2001/03/webdata/xsv`
- submit XML file (and schema file)
- report generated for `greeting.xml` as follows

More complex document example

```
<cd xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="cd.xsd">
  <composer>Johannes Brahms</composer>
  <performance>
    <composition>Piano Concerto No. 2</composition>
    <soloist>Emil Gilels</soloist>
    <orchestra>Berlin Philharmonic</orchestra>
    <conductor>Eugen Jochum</conductor>
    <recorded>1972</recorded>
  </performance>
  <performance>
    <composition>Fantasias Op. 116</composition>
    <soloist>Emil Gilels</soloist>
    <recorded>1976</recorded>
  </performance>
  <length>PT1H13M37S</length>
</cd>
```

Simple and complex data types

- XML schema allows definition of *data types* as well as declarations of elements and attributes
- simple data types
  - can contain only text (i.e., no markup)
  - e.g., values of attributes
  - e.g., elements without children or attributes
- complex data types can contain
  - child elements (i.e., markup) or
  - attributes

More complex schema example

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

  <xsd:element name="cd" type="CDType"/>

  <xsd:complexType name="CDType">
    <xsd:sequence>
      <xsd:element name="composer"    type="xsd:string"/>
      <xsd:element name="performance" type="PerfType"
                                      maxOccurs="unbounded"/>
      <xsd:element name="length"      type="xsd:duration"
                                      minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
  ...
</xsd:schema>
```

Main schema components

- `xsd:element` *declares* an element and assigns it a type, e.g.,

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

  using a built-in, simple data type, or

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

  using a user-defined, complex data type
- `xsd:complexType` *defines* a new type, e.g.,

  ```
  <xsd:complexType name="CDType">
  ...
  </xsd:complexType>
  ```

- defining named types allows reuse (and may help readability)
- `xsd:attribute` *declares* an attribute and assigns it a type (see later)

Structuring element declarations

- `xsd:sequence`
  - requires elements to occur in order given
  - analogous to , in DTDs
- `xsd:choice`
  - allows one of the given elements to occur
  - analogous to | in DTDs
- `xsd:all`
  - allows elements to occur in any order
  - analogous to & in *SGML* DTDs

Defining number of element occurrences

- `minOccurs` and `maxOccurs` attributes control the number of occurrences of an element, sequence or choice
- `minOccurs` must be a non-negative integer
- `maxOccurs` must be a non-negative integer or `unbounded`
- default value for each of `minOccurs` and `maxOccurs` is 1

Another complex type example

```
<xsd:complexType name="PerfType">
  <xsd:sequence>
    <xsd:element name="composition" type="xsd:string"/>
    <xsd:element name="soloist"     type="xsd:string"
                                    minOccurs="0"/>
    <xsd:sequence minOccurs="0">
      <xsd:element name="orchestra" type="xsd:string"/>
      <xsd:element name="conductor" type="xsd:string"/>
    </xsd:sequence>
    <xsd:element name="recorded"    type="xsd:gYear"/>
  </xsd:sequence>
</xsd:complexType>
```

An equivalent DTD

```
<!ELEMENT CD          (composer, (performance)+, (length)?)>
<!ELEMENT performance (composition, (soloist)?,
                       (orchestra, conductor)?, recorded)>
<!ELEMENT composer    (#PCDATA)>
<!ELEMENT length      (#PCDATA)> <!-- duration -->
<!ELEMENT composition (#PCDATA)>
<!ELEMENT soloist     (#PCDATA)>
<!ELEMENT orchestra   (#PCDATA)>
<!ELEMENT conductor   (#PCDATA)>
<!ELEMENT recorded    (#PCDATA)> <!-- gYear -->
```

Declaring attributes

- use `xsd:attribute` element inside an `xsd:complexType`
- has attributes `name`, `type`, e.g.,

```
<xsd:attribute name="version" type="xsd:number"/>
```

- attribute `use` is optional
  - if omitted means attribute is optional (like `#IMPLIED`)
  - for required attributes, say `use="required"` (like `#REQUIRED`)
- for fixed attributes, say `fixed="..."` (like `#FIXED`), e.g.,

```
<xs:attribute name="version" type="xs:number" fixed="2.0"/>
```

- for attributes with default value, say `default="..."`
- for enumeration, use `xsd:simpleType`
- attributes must be declared at the end of an `xsd:complexType`

Locally-scoped element names

- in DTDs, all element names are *global*
- XML schema allows element types to be local to a context, e.g.,

```
<xsd:element name="book">
      <xsd:element name="title"> ... </xsd:element>
    ...
</xsd:element>

<xsd:element name="employee">
      <xsd:element name="title"> ... </xsd:element>
    ...
</xsd:element>
```

- content models for two occurrences of `title` can be different

Simple data types

- Form a type hierarchy; the root is called *anyType*
  - all complex types
  - *anySimpleType*
    - `string`
    - `boolean`, e.g., `true`
    - `anyUri`, e.g., `http://www.dcs.bbk.ac.uk/~ptw/home.html`
    - `duration`, e.g., `P1Y2M3DT10H5M49.3S`
    - `gYear`, e.g., `1972`
    - `float`, e.g., `123E99`
    - `decimal`, e.g., `123456.789`
    - ...
- lowest level above are the *primitive data types*
- for a full list, see Simple Types in the Primer

Primitive date and time types

- `date`, e.g., `1994-04-27`
- `time`, e.g., `16:50:00+01:00` or `15:50:00Z` if in Co-ordinated Universal Time (UTC)
- `dateTime`, e.g., `1918-11-11T11:00:00.000+01:00`
- `duration`, e.g., `P2Y1M3DT20H30M31.4159S`
- "Gregorian" calendar dates (introduced in 1582 by Pope Gregory XIII):
  - `gYear`, e.g., `2001`
  - `gYearMonth`, e.g., `2001-01`
  - `gMonthDay`, e.g., `-12-25` (note hyphen for missing year)
  - `gMonth`, e.g., `-12-` (note hyphens for missing year and day)
  - `gDay`, e.g., `--25` (note only 3 hyphens)

Built-in derived string types

Derived from `string`:

- `normalizedString` (newline, tab, carriage-return are converted to spaces)
  - `token` (adjacent spaces collapsed to a single space; leading and trailing spaces removed)
    - `language`, e.g., `en`
    - `name`, e.g., `my:name`

Derived from `name`:

- `NCNAME` ("non-colonized" name), e.g., `myName`
  - `ID`
  - `IDREF`
  - `ENTITY`

Built-in derived numeric types

Derived from `decimal`:

- `integer` (decimal with no fractional part), e.g., -123456
  - `nonPositiveInteger`, e.g., 0, -1
    - `negativeInteger`, e.g., -1
  - `nonNegativeInteger`, e.g., 0, 1
    - `positiveInteger`, e.g., 1
    - `...`
  - `...`

User-derived simple data types

- complex data types can be created "from scratch"
- new simple data types must be *derived* from existing simple data types
- derivation can be by one of
  - *extension*
    - *list*: a list of values of an existing data type
    - *union*: allows values from two or more data types
  - *restriction*: limits the values allowed using, e.g.,
    - maximum value (e.g., 100)
    - minimum value (e.g., 50)
    - length (e.g., of string or list)
    - number of digits
    - enumeration (list of values)
    - pattern

  above constraints are known as *facets*

Restriction by enumeration

```
<xsd:element name="MScResult">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="distinction"/>
      <xsd:enumeration value="merit"/>
      <xsd:enumeration value="pass"/>
      <xsd:enumeration value="fail"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

- contents of `MScResult` element is a restriction of the `xsd:string` type
- must be one of the 4 values given
- e.g., `<MScResult>pass</MScResult>`

Restriction by values

- `examMark` can be from 0 to 100

```
<xsd:element name="examMark">
  <xsd:simpleType>
    <xsd:restriction base="xsd:nonNegativeInteger">
      <xsd:maxInclusive value="100"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

- or, equivalently

```
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="100"/>
    </xsd:restriction>
```

Restriction by pattern

```
<xsd:element name="zipcode">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{5}(-\d{4})?"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

- `value` attribute contains a *regular expression*
- `\d` means any digit
- `()` used for grouping
- `x{5}` means exactly 5 `x`'s (in a row)
- `x?` indicates zero or one `x`
- zipcode examples: 90720-1314 and 22043

Document with mixed content

- We may want to mix elements and text, e.g.:

```
<letter>
  Dear Mr <name>Smith</name>,
  Your order of <quantity>1</quantity>
  <product>Baby Monitor</product> was shipped
  on <date>1999-05-21</date>. ....
</letter>
```

- A DTD would have to contain:

```
<!ELEMENT letter (#PCDATA|name|quantity|product|date)*>
```

which cannot enforce the order of subelements

Schema fragment declaring mixed content

```
<xsd:element name="letter">
 <xsd:complexType mixed="true">
  <xsd:sequence>
   <xsd:element name="name" type="xsd:string"/>
   <xsd:element name="quantity" type="xsd:positiveInteger"/>
   <xsd:element name="product"  type="xsd:string"/>
   <xsd:element name="date" type="xsd:date" minOccurs="0"/>
   <!-- etc. -->
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

Chapter 5

Relax NG

Problems with DTDs

- DTDs are sometimes not powerful enough
- e.g., (to simplify) in HTML
  1. a `form` element can occur in a `table` element and
  2. a `table` element can occur in a `form` element, but
  3. a `form` element *cannot* occur inside another `form` element
- we have

```
<!ELEMENT table  (... form ...) >
<!ELEMENT form   (... table ...) >
```

- but condition (3) above cannot be enforced by an XML DTD

Problems with XML schema

- XML schema *can* handle the previous example using locally-scoped element names
- but what about the following?
  - a document (`doc` element) contains one or more paragraphs (`par` elements)
  - the first paragraph has a different content model to subsequent paragraphs
  - (perhaps the first letter of the first paragraph is enlarged)
- we want something like

```
<!ELEMENT doc  (par, par*) >
```

  but where two occurrences of `par` have *different* content models
- this *cannot* be specified in XML schema

RelaxNG

- RelaxNG resulted from the merger of two earlier projects
  - RELAX (REgular LAnguage description for XML)
  - TREX (Tree Regular Expressions for XML)
- It has the same power as *Regular Tree Grammars*
- It has two syntactic forms: one XML-based, one not (called the *compact* syntax)
- It is simpler than XML schema
- It uses XML Schema Part 2 for a vocabulary of data types

Compact Syntax: RSS Example

```
element rss {
  element channel {
    element title          { text },
    element link           { xsd:anyURI },
    element description    { text }.
    element lastBuildDate { xsd:dateTime }?,
    element ttl            { text }?,
    element item {
      element title        { text },
      element description { text },
      element link         { xsd:anyURI }?
      element pubDate      { xsd:dateTime }?
    }+
  }
}
```

Named patterns

- It is often convenient to be able to give *names* to parts of a pattern
- This is similar to using *non-terminal* symbols in a (context-free) grammar
- It is also related to the use of complex types in XSDL
- RelaxNG uses “=” in the compact syntax (and `define` elements in the XML syntax) to give names to patterns
- The name `start` is used for the root pattern

Compact Syntax with Named Patterns: RSS Example

```
start   = RSS
RSS     = element rss     { Channel }
Channel = element channel { Title,Link,Desc,LBD?,TTL?,Item+ }
Title   = element title   { text },
Link    = element link    { xsd:anyURI },
Desc    = element description   { text },
LBD     = element lastBuildDate { xsd:dateTime },
TTL     = element ttl           { text },
Item    = element item          { Title, Desc, Link, PD? }
PD      = element pubDate       { xsd:dateTime }
```

Table and forms example (compact syntax)

```
TableWithForm    = element table { ... Form ... }

Form             = element form  { ... TableWithoutForm ... }

TableWithoutForm = element table { ... }
```

- No `Form` pattern appears in the third definition above

Paragraphs example (compact syntax)

```
D  = element doc { P1, P2* }

P1 = element par { ... }

P2 = element par { ... }
```

- The content models for the `P1` and `P2` patterns can be different

Summary

- We have considered 3 different languages for defining XML document types
- DTDs are simple, but their main limitation is that data types (other than strings) are not provided
- XSDL is comprehensive, but rather complicated
- RelaxNG is the most expressive of the three, while still remaining quite simple; it is also an ISO standard, but has not been widely adopted