Modelling XML Applications (part 2)

Patryk Czarnik

XML and Applications 2013/2014
Lecture 3 – 21.10.2013

Modularisation options

- Combining multiple files
  - DTD – external parameter entities
  - Schema – include, import, redefine
- Reusing fragments of model definition
  - DTD – parameter entities
  - Schema – groups and attribute groups
    (in practice equivalent to the above)
  - Schema – types, type derivation (no such feature in DTD)
- Global and local definitions
  - In DTD all elements global, all attributes local
  - In schema both can be global or local, depending on case

See examples for details!

Import or include?

- `xs:import`
  - Imports foreign definitions, enables referring to them
- `xs:redefine`
  - Includes external definitions, but a local definition overrides external one if they share the same name
- `xs:include`
  - Basic command, almost like textual insertion
  - Imported module must have the same target namespace or no target namespace

A multi-module, namespace-aware project with overused `xs:include` leads to duplication of logic in the software that processes documents (or enforces meta-programming tricks to avoid it). */based on personal experience/*

Schema and namespaces

- DTD is namespace-ignorant
- XML Schema conceptually and technically bound with XML namespaces
  - Basic approach: one schema (file) = one namespace
    - It is also possible to split one ns into several files
  - Referring to components from other namespaces available
- Important attributes
  - `targetNamespace` – if given, all global definitions within a schema go into that namespace
  - `elementFormDefault`, `attributeFormDefault` – should local elements or attributes have qualified names?
    - default for both: `unqualified`
    - typical approach: elements qualified, attributes unqualified
    - setting may be changed for individual definitions

Using namespaces in XML Schema

Different technical approaches to handle namespaces in XML Schema

- XML Schema ns. bound to `xs:` or `xsd:`, no target namespace
- XML Schema ns. bound to `xs:` or `xsd:`, target namespace as default namespace
  - Convenient as long as we don't use keys and keyrefs
- Target namespace bound to a prefix (`tns:` by convention)
- Then we can declare XML Schema as default namespace and avoid using `xs:` or `xsd:`

Types in XML Schema

- Every element and attribute has a type
  - If not specified: xs:anyType or xs:anySimpleType, resp.
- “What an element/attribute may contain”
  but also
  “How to interpret a value”

Classification of types

Types by content model

- Simple type (value of a text node or an attribute; applicable to elements and attributes)
  - atomic type
  - list
  - union
- Complex type (structure model – subelements and attributes; applicable to elements)
  - empty content
  - element content
  - mixed content
  - simple content

Classification of types

Types by place of definition:

- anonymous – defined locally in place of use
- named – defined globally
  - built-in – defined in XML Schema specification
  - user-defined

Types by means of definition:

- primitive (simple types)
- defined directly (complex type as a sequence etc.)
- derived (some built-in types are defined by derivation!)
  - by extension (complex types only)
  - by restriction (complex and simple types)
  - as a list or union (simple types only)

Simple types

- Rich set of built-in types
  - decimal, integer, nonNegativeInteger, long, int, ...
  - boolean, float, double
  - date, time, dateTime, duration, ...
  - string, token, base64Binary, hexBinary, ...
  - See the recommendation for the complete hierarchy
- Defining custom types basing on built-in types
  - by restriction
  - as a list
  - as an union

Value space vs lexical space

- A simple type specifies its
  - **value space** – set of abstract values
  - **lexical space** – set of valid text representations

Type	Text representations	Abstract value
xs:boolean	`0`, `false` `1`, `true`	False True
xs:decimal (and derivatives)	`13`, `013`, `13.00`	13
xs:string	`013` `  foo  bar  `	'013' ' foo bar '
xs:token	`  foo  bar  `	'foo bar'

Choosing the appropriate type

- Semantic meaning of a simple type:
  - not only a “set of allowed character strings”
  - also the way a value is interpreted!
- Types may affect the validation
  - e.g. leading zeros significant in strings, meaningless in numbers
- Processors may use the information about type, e.g.
  - schema-aware processing in XSLT 2.0 or XQuery
    - sorting, comparison, arithmetic operations
  - JAXB – generation of Java classes based on XSD
- Choosing the appropriate type sometimes not obvious
  - phone number, zip code, room number – number or string?

Defining simple types by restriction

- Constraining facets – properties we can restrict
  - enumeration
  - pattern
  - length, minLength, maxLength
  - totalDigits, fractionDigits
  - maxInclusive, maxExclusive
  - minInclusive, minExclusive
  - whiteSpace

Some of them available only for chosen primitive base types

- Used directly in simple type definition:

```
<xs:simpleType name="LottoNumber">
   <xs:restriction base="xs:integer">
      <xs:minInclusive value="1" />
      <xs:maxInclusive value="49"/>
   </xs:restriction>
</xs:simpleType>
```

```
<lottoNumber>12</lottoNumber>
```

List types

- List of values separated with whitespace.
- Not to confuse with sequences
  - list – simple type, no markup structure within
  - sequence – complex type, sequence of subelements
- Compact notation for lists of values

but

- Harder to process in XML processors (requires additional parsing using regexp etc. – not available e.g. in XSLT 1.0)

```
<xs:simpleType name="LottoNumberList">
   <xs:list itemType="LottoNumber" />
</xs:simpleType>
```

```
<lottoNumberList>12 2 47 6 33 12 27 18</lottoNumberList>
```

Union types

- Union of sets of values
- Possibility to mix values of different primitive types
  - Interpreting values as abstract values hard to perform
  - Nevertheless, a usable feature (e.g. unbounded in XML Schema)

```
<size>40</size>
<size>L</size>
```

```
<xs:simpleType name="ClothingSizeLetter">
  <xs:restriction base="xs:token">
    <xs:enumeration value="XS"/>
    <xs:enumeration value="S" />
    <xs:enumeration value="M" />
    <xs:enumeration value="L" />
    <xs:enumeration value="XL"/>
    <xs:enumeration value="XXL"/>
  </xs:restriction>
</xs:simpleType>
```

```
<xs:simpleType name="ClothingSizeNumber">
   <xs:restriction base="xs:integer">
      <xs:minInclusive value="20" />
      <xs:maxInclusive value="60" />
   </xs:restriction>
</xs:simpleType>
```

```
<xs:simpleType name="ClothingSize">
   <xs:union memberTypes="ClothingSizeNumber ClothingSizeLetter"/>
</xs:simpleType>
```

Identity constraints

- Constraints on uniqueness and references

Two mechanisms:

- DTD attribute types `ID` and `IDREF`
  - introduced in SGML DTD but still available in XML Schema
  - drawbacks:
    - one global scope, at most one ID per element
    - special form of values – only names allowed
    - IDs and references necessarily in attributes
- XML Schema identity constraints
  - `key`, `unique`, and `keyref` definitions
  - more powerful and more flexible than ID/IDREF