XML and Content Management

Lecture 5: Patterns in schema design

Maciej Ogrodniczuk

MIMUW, 8 November 2010

Good common practice of schema design

1. **readability**: schemata which are easy to understand are easier to maintain and they have higher chances to be often used and reused,
2. **precision**: properly constructed types can eliminate errors in data (important for exchanging data with applications out of our control),
3. **reuse**: usually means savings and better design, „less means more",
4. **flexibility and extensibility**: helps satisfying various user requirements and further schema reuse, supports change management.

10 concrete hints

By Priscilla Walmsley:

1. which names to use?
2. specific or general names of properties?
3. how to resolve name conflicts?
4. `xsd:string`, `xsd:normalizedString` or `xsd:token`?
5. grouping elements?
6. nil values,
7. lists of values,
8. namespaces,
9. global or local declaration of elements?
10. named or anonymous types?

http://www.datypic.com/services/xmldesign/XMLDesignColor.pdf

Names of schema components

Names should be:

- meaningful and easy to remember,
- not too much magical or abbrieviated (PTNM),
- handy (toyShopConstructionSiteAddress),
- used consistently (ZipCode, zip-code, zip_code, zip.code, zipCode), with standardized vocabulary (zipCode, postCode, postalCode, kodPocztowy),
- prefixed or suffixed by types and groups.

`<product>/<productNumber>` or `<product>/<number>`?

+ more obvious relation to the meaning of the element,
+ easier processing (indepedent of the parent element, so that the child name can be used to retrieve it),
− too explicit,
− hides similarity between numbers in different elements.

Specific or general names of properties?

Specific names:

```
<length>60</length>
<width>50</width>
<height>52</height>
<weight>25</weight>
```

+ types can be assigned,
+ optionality and occurrence can be defined,
− schema must be changed when new property is introduced.

General names:

```
<property name="length">
          60</property>
<property name="width">
          50</property>
<property name="height">
          52</property>
<property name="weight">
          25</property>
```

+ schema does not change when new property appears,
+ easier processing (e.g. displaying all properties),
− types, optionality and occurrence cannot be defined.

Name conflicts

Is such definition correct?

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="text"/>
  <xsd:element name="text"/>
  <xsd:attribute name="text"/>
  <xsd:simpleType name="text">
    <xsd:list itemType="xsd:token"/>
  </xsd:simpleType>
</xsd:schema>
```

General rule:

- elements and attributes can have the same names,
- types can have the same names as elements or attributes,
- two types cannot have the same name.

xsd:string, xsd:normalizedString or xsd:token?

xsd:string

- when whitespace formatting is important,
- for long texts — using mixed content can be also considered.

xsd:normalizedString

- when whitespace formatting is not important,
  but positions of characters are,

xsd:token

- best for short texts, particularly those restricted
  by enumeration or pattern.

Grouping elements

Without a grouping element:

```
<customer>
  <name>MIMUW</name>
  <street>Banacha 2</street>
  <zipCode>02-097</zipCode>
  <city>Warsaw</city>
</customer>
```

With a grouping element:

```
<customer>
  <name>MIMUW</name>
  <address>
    <street>Banacha 2</street>
    <zipCode>02-097</zipCode>
    <city>Warsaw</city>
  </address>
</customer>
```

+ more intuitive, more easily filled in by people,
+ easier to process, e.g. by XSLT,
− a little too verbose?

Nil values

Using *nil values*:

- does not impair the definition by allowing empty content,
- allows unambiguous definition that the information does not exist,
- allows passing information about indeterminacy without removing the element from content (presence of the element may be used by the application),
- allows turning off default values.

Nil values cannot be used for non-string values, unless...
some trick is applied:

```
<xsd:simpleType>
  <xsd:union memberTypes="xsd:integer">
    <xsd:simpleType>
      <xsd:restriction base="xsd:token">
        <xsd:enumeration value=""/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>
```

Lists of values

An example:

```
<xsd:simpleType name="USStateType">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="Alabama"/>
    <xsd:enumeration value="Alaska"/>
    ...
  </xsd:restriction>
</xsd:simpleType>
```

Challenges:

1. potentially frequent changes, often out of control of the schema designer (language, currency, country codes) → worth defining in separate schema documents which allows versioning,
2. length of lists slowing down validation, littering the schema, hindering schema management → worth limiting list length to 15-20 values, document the lists, use patterns,
3. inextensibility of the simple types → `xsd:token` or its union with enumeration can be used to let other documents narrow the list.

Extensible list of values: an example

```
<xsd:simpleType name="canadianLanguagesType">
  <xsd:union memberTypes="xsd:token">
    <xsd:simpleType>
      <xsd:restriction base="xsd:token">
        <xsd:enumeration value="English"/>
        <xsd:enumeration value="French"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>

<xsd:simpleType name="canadianLanguagesAfterImmigrationType">
  <xsd:restriction base="canadianLanguages">
    <xsd:enumeration value="Polish"/>
    <xsd:enumeration value="English"/>
  </xsd:restriction>
</xsd:simpleType>
```

Namespaces

General ideas:

- namespaces should be declared in root element,
- default namespace should be used only when it dominates the document,
- prefixes do not matter, but conventions should be applied (e.g. `xsd` or `xs` for schemata...)

3 ways of using namespaces in the schema:

1. target namespace as default namespace (most frequent solution),
2. XML Schema namespace as default,
3. no default namespace.

Default target namespace

Notation:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://mimuw.edu.pl/usos"
            xmlns="http://mimuw.edu.pl/usos">
  <xsd:complexType name="personType">
  ...
  </xsd:complexType>
  <xsd:element name="student" type="personType"/>
  <xsd:element name="studies-start-date" type="xsd:date"/>
</xsd:schema>
```

Notes:

- XML Schema components are always prefixed,
- references to components defined in the schema without the prefix (apart from XPaths in `<xsd:key>` and `<xsd:keyref>` elements – default namespace does not function there).

Default XML Schema namespace

Notation:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://mimuw.edu.pl/usos"
        xmlns:usos="http://mimuw.edu.pl/usos">
  <complexType name="personType">
  ...
  </complexType>
  <element name="student" type="usos:personType"/>
  <element name="studies-start-date" type="date"/>
</schema>
```

Notes:

- XML Schema components are not prefixed,
- references to components defined in the schema are prefixed.

BUT: target namespace necessary to reference defined components!

No default namespace

Notation:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://mimuw.edu.pl/usos"
            xmlns:usos="http://mimuw.edu.pl/usos">
  <xsd:complexType name="personType">
  ...
  </xsd:complexType>
  <xsd:element name="student" type="usos:personType"/>
  <xsd:element name="studies-start-date" type="xsd:date"/>
</xsd:schema>
```

Notes:

- every reference with a prefix (apart from new names),
- best solution while having many namespaces.

Global or local element declarations?

Global declarations:

```
<xsd:element name="person"/>
<xsd:element name="winners">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="person"
           maxOccurs="10"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

+ can be reused,
− must have unique names in the whole schema (with all attached parts!)

Local declarations:

```
<xsd:element name="winners">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="person"
           maxOccurs="10"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

+ do not have to be unique in the schema,
− cannot be validated independently of the main document.

Named or anonymous types?

Named types:

```
<xsd:complexType
     name="from1to10people">
  <xsd:sequence>
    <xsd:element name="person"
         maxOccurs="10"/>
  </xsd:sequence>
</xsd:complexType>
```

- + can be reused,
- + can be used for derivation or in list and union definitions,
- + make complex schemata more readable.

Anonymous types:

```
<xsd:element name="winners">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="person"
           maxOccurs="10"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

- + can simplify schema maintenance (if the schema is not reused) since they don't influence other components,
- + can be more readable in simple cases.

3 methods of schema documentation

Using:

1. a dedicated element `<xsd:annotation>`,
2. XML comments,
3. *foreign attributes*.

or in non-schema-related ways:

- describing the schema outside it,
- storing the schema together with descriptions of components, allowed transformations and styles – in any format,
- ...

<xsd:annotation> — schema documentation

<xsd:annotation> element can be used in any place on the global level or at the beginning of any XML Schema construct.

Its content is made of <xsd:documentation> and <xsd:appinfo> elements carrying additional information (text and tags) respectively „for people and computers".

```
<xsd:element name="surname" type="xsd:string">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      Addressee's last name.
    </xsd:documentation>
    <xsd:appinfo>
      <form:label>Surname of the recipient:</form:label>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
```

<xsd:appinfo> definition

```
<xsd:element name="appinfo">
  <xsd:annotation>
    <xsd:documentation source="http://www.w3.org/TR/
                         xmlschema-1/#element-appinfo"/>
  </xsd:annotation>
  <xsd:complexType mixed="true">
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
      <xsd:any processContents="lax"/>
    </xsd:sequence>
    <xsd:attribute name="source" type="xsd:anyURI"/>
  </xsd:complexType>
</xsd:element>
```

Both elements can carry an optional `source` attribute documenting the source of information.

What is really <xsd:appinfo> for?

Can be used for:

- storing additional validation rules (e.g. for Schematron),
- mapping to other technologies (e.g. relational schema),
- mapping to forms (e.g. XHTML <form> tags).

Foreign attributes

Every schema component can carry attributes coming from any namespace. Such attributes are not validated and do not have to be declared.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:doc="http://www.mimuw.edu.pl/xml">
  <xsd:element name="lecture" type="presentation"
               doc:desc="Lecture presentation
                         root element."/>
</xsd:schema>
```

Managing complex schemata

In complex environments standarization should be used for:

- naming,
- methods of schema construction,
- methods of schema extension,
- scope of using XML Schema constructs.

Such recommendations are often formally described in an NDR (*naming and design rules*) document.

Software should be used to control:

- edition: who is changing what,
- versioning schema components,
- schema fragments used in different applications.

More on the process of modelling

How?

- analyzing dependencies between modelled objects and their parts,
- isolating substructures,
- analyzing available sample documents,
- analyzing potential applications and use cases,
- building abstract structure design,
- writing the model down,
- testing it in "the real world",
- maintaining the structure while using it,
- managing the change.

Structure change management

The problem:

- The structure needs to be changed.

Solution variants:

1. the change is made backward compatible (e.g. only optional elements are added),
2. two versions of the schema are used,
3. documents are migrated: automatically and/or telling users to do it.

Making applications change-resistant

A few methods:

- moving document validation to schema level,
- neglecting unimportant elements and attributes,
- avoiding structure dependance: `//product/number` instead of `/catalogue/product/number`,
- schema parametrization — using fixed attributes, to store form field names, database column names etc.,
- using namespaces (see XLink standard).

Rule of thumb: common sense.

What has not been mentioned (and won't be)

More topics for self-study:

- dependency of facets and data types,
- validation rules and schema constraints,
- using entities, notations, complete list of less-frequently used DTD equivalents,
- *substitution groups*,
- schema structure models (*Russian Doll*, *Salami Slice*, *Venetian Blind*, *Garden of Eden*),
- full XML Schema specification.

See also Priscilla Walmsley book *Definitive XML Schema*, Prentice Hall (*Wszystko o XML Schema*, WNT 2008).
Examples from the book: `http://www.datypic.com/books/defxmlschema/examples.html`.

XML Schema criticism

Summary by James Clark:

- A schema written using XSD is difficult to read and understand.
- There are many surprises in the language, for example that restriction of elements works differently from restriction of attributes.
- The W3C Recommendation itself is extremely difficult to read.
- XSD lacks any formal mathematical specification.
- XSD provides no facilities to state that the value or presence of one attribute is dependent on the values or presence of other attributes (so-called co-occurrence constraints).
- XSD offers very weak support for unordered content.
- The set of XSD datatypes on offer is highly arbitrary.
- There is no way for an XSD schema to indicate which elements are permitted at the top level of a document.
- ...

What cannot be done in XML Schema?

The problem:

- context-dependent validation,
  e.g. *<net-price> element value is lower than or equal to <gross-price> element value*,
  or: *list of lottery numbers is sorted.*

Solutions:

- program new constraints in the application code,
- use XSLT,
- use other schema language, e.g. Schematron.

What else?

Problems:

- *ambiguity*, i.e. conformance of the fragment to many patterns,
- *non-determinism*, when document processor may select many patterns (grammar productions) and there is no equivalent deterministic model.

Solution:

- Relax NG = REgular LAnguage description for XML – New Generation.

Schematron

By Rick Jelliffe (1999). ISO standard since 2006.
(as part 3 of DSDL standard — Document Schema Definition Languages).

Idea: `<pattern>`s containing context validation `<rule>`s
composed of XPath-enabled required `<assert>`ions
and `<report>`s (describing errors).

Schematron schema for schematron (`www.schematron.com`):

```
<schema xmlns="http://www.mimuw.edu.pl/dsdl/schematron">
  <ns prefix="sch" uri="http://www.mimuw.edu.pl/dsdl/schematron">
  <pattern>
    <rule context="sch:schema">
      <assert test="sch:pattern">Schema is composed of patterns.</assert>
      <assert test="sch:pattern/sch:rule[@context]">Each pattern
          is composed of rules carrying 'context' attribute.</assert>
      <assert test="sch:pattern/sch:rule/sch:assert[@test]
           or sch:pattern/sch:rule/sch:report[@test]">A rule
           is composed of 'assert' and 'report' instructions
           carrying 'test' attribute.</assert>
    </rule>
  </pattern>
</schema>
```

Schema built-in assertions: an example

```
<xsd:element name="product">
  <xsd:annotation>
    <xsd:appinfo>
      <sch:pattern name="Gross price higher than net price.">
        <sch:rule context="shop:product">
          <sch:assert test="shop:grossPrice > shop:netPrice">
             Number value of 'grossPrice' element should be larger
             than number value of 'netPrice' element.</sch:assert>
        </sch:rule>
      </sch:pattern>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="netPrice" type="xsd:integer"/>
      <xsd:element name="grossPrice" type="xsd:integer"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Relax NG in a nutshell

By OASIS, based on RELAX by Makoto Murata and TREX by James Clark.

ISO standard since 2006 — as part 2 of DSDL standard.

Regular Language description for XML – New Generation:

- two syntax variants: XML and „compact",
- namespace support,
- uniform treatment of elements and attributes,
- support for unordered and mixed content,
- may be used with a separate type language (e.g. XML Schema).

Mostly benefits:

+ simpler schema description,
+ technically more advanced,
+ offering more possibilities of expression,
− far less popular (= less supported by software companies).

Relax NG and DTD

RELAX NG extends DTD functionality by:

- introducing data types,
- integrating attributes with content model,
- supporting namespaces,
- supporting any order of elements,
- supporting context models.

At the same time RELAX NG:

- does not validate `ID/IDREF`
  (but there is an enhancement removing this problem),
- does not allow default attributes,
- does not allow character entities, notations,
- does not allow specification of whitespace preservation method,
- does not define the way of binding Relax NG schema
  with the document.

Two variants of RELAX NG syntax

DTD:

```
<!ELEMENT business-cards
          (business-card*)>
<!ELEMENT business-card
          (person, e-mail)>
<!ELEMENT person (#PCDATA)>
<!ELEMENT e-mail (#PCDATA)>
```

Short syntax:

```
element business-cards {
  element business-card {
    element person { text }},
    element e-mail { text }
  }*
}
```

http://www.relaxng.org

XML syntax:

```
<element
 name="business-cards"
 xmlns="http://relaxng.org/
        ns/structure/1.0">
  <zeroOrMore>
    <element
      name="business-card">
      <element name="person">
        <text/>
      </element>
      <element name="e-mail">
        <text/>
      </element>
    </element>
  </zeroOrMore>
</element>
```

RELAX NG tutorial: required element

RELAX NG:

```
<element name="business-cards"
         xmlns="http://relaxng.org/ns/structure/1.0">
  <oneOrMore>
    <element name="business-card">
    ...
    </element>
  </oneOrMore>
</element>
```

DTD:

```
<!ELEMENT business-cards (business-card+)>
```

RELAX NG tutorial: optional element

RELAX NG:

```
<element name="business-card">
  <element name="person">
    <text/>
  </element>
  <element name="e-mail">
    <text/>
  </element>
  <optional>
    <element name="phone-number">
      <text/>
    </element>
  </optional>
</element>
```

DTD:

```
<!ELEMENT business-card (person, e-mail, phone-number?)>
```

RELAX NG tutorial: group and choice

RELAX NG:

```
<element name="business-card">
  <choice>
    <element name="person">
      <text/>
    </element>
    <group>
      <element name="first-name">
        <text/>
      </element>
      <element name="surname">
        <text/>
      </element>
    </group>
  </choice>
</element>
```

DTD:

```
<!ELEMENT business-card (person | (first-name, surname))>
```

RELAX NG tutorial: empty element and attributes

RELAX NG:

```
<element name="person">
  <attribute name="phone-number"/>
</element>
```

Note:

- `<text/>` is default attribute content,
- attributes are required by default, `<optional>` must be used to mean `IMPLIED`,
- when there are no attributes, empty elements are marked `<empty/>`.

DTD:

```
<!ELEMENT person EMPTY>
<!ATTLIST person phone-number CDATA REQUIRED>
```

RELAX NG tutorial: fragments and *doctype*

RELAX NG:

```
<grammar>
  <start>
    <element name="business-cards">
      <zeroOrMore>
        <element name="business-card">
          <ref name="card"/>
        </element>
      </zeroOrMore>
    </element>
  </start>

  <define name="card">
    <element name="person">
      <text/>
    </element>
    <element name="e-mail">
      <text/>
    </element>
  </define>
</grammar>
```

DTD:

```
<!DOCTYPE
  business-cards

  [<!ELEMENT
     business-cards
     (business-card*)>

   <!ENTITY %
     card
     "person, e-mail">

   <!ELEMENT
     business-card
     (%card;)>

   <!ELEMENT
     person
     (#PCDATA)>

   <!ELEMENT
     e-mail
     (#PCDATA)>]
```

RELAX NG tutorial: data types

RELAX NG:

```
<element name="e-mail" datatypeLibrary=
         "http://www.w3.org/2001/XMLSchema-datatypes">
  <data type="string">
    <param name="maxLength">127</param>
  </data>
</element>
```

XML Schema:

```
<xsd:element name="e-mail">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="127"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

RELAX NG tutorial: enumerations

RELAX NG:

```
<element name="file">
  <attribute name="format">
    <choice>
      <value>HTML</value>
      <value>PDF</value>
    </choice>
  </attribute>
</element>
```

Element content can be enumerated as well (as in XML Schema).

DTD:

```
<!ATTLIST file format (HTML | PDF) #REQUIRED>
```

RELAX NG tutorial: lists

RELAX NG:

```
<element name="pairs">
  <list>
    <oneOrMore>
      <data type="integer"/>
      <data type="integer"/>
    </oneOrMore>
  </list>
</element>
```

RELAX NG tutorial: any order, mixed content

RELAX NG:

```
<element name="head">
  <interleave>
    <ref name="title"/>
    <zeroOrMore>
      <ref name="meta"/>
    </zeroOrMore>
  </interleave>
</element>

<element name="p">
  <mixed>
    <ref name="b"/>
    <ref name="i"/>
    <ref name="u"/>
  </mixed>
</element>
```

DTD for mixed content model:

```
<!ELEMENT p
  (#PCDATA | b | i | u)*>
```

RELAX NG: altogether

- `<element name="`*value*`">`, `<attribute name="`*value*`">`,
- `<optional>` — optional attribute or element (attributes required by default),
- `<text/>` — means `#PCDATA`,
- `<empty/>` — empty element,
- `<zeroOrMore>`, `<oneOrMore>`,
- `<choice>`, `<group>`, `<interleave>` (any order),
- `<mixed>`,
- `<grammar>` — root element of the grammar,
- `<start>` — root element of the schema,
- ...

RELAX NG: simple or element content

```
<element name="person">
  <choice>
    <text/>
    <group>
      <element name="first-name">
        <data type="token"/>
      </element>
      <optional>
        <element name="second-name">
          <data type="token"/>
        </element>
      </optional>
      <element name="last-name">
        <data type="token"/>
      </element>
    </group>
  </choice>
</name>
```

Examplotron

By Eric van der Vlist; work started in 2001, suspended in 2003.

Idea: „definition by example" – instances of compliant documents define the schema.

How does it work? XSLT transformation converts instances into „real" RELAX NG schema.

80% cases are simple:

- definition of elements and attributes,
- controlling occurrence,
- mixed content,
- predefined simple XML Schema types.

`http://examplotron.org`

Examplotron: an example

Examplotron:

```
<person>
  <name>Daisy</name>
  <name>Rachel</name>
</person>
```

Relax NG:

```
<grammar xmlns:ega="http://examplotron.org/annotations/"
   xmlns="http://relaxng.org/ns/structure/1.0"
   xmlns:sch="http://www.ascc.net/xml/schematron"
   datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <start>
    <element name="person">
      <oneOrMore>
        <element name="name">
          <text><ega:example>Daisy</ega:example>/text>
        </element>
      </oneOrMore>
    </element>
  </start>
</grammar>
```