XML Technologies
Validation

Christian Grün
Database & Information Systems Group
Universität Konstanz

Introduction

- in *relational databases*, data has fixed *types* (`varchar`, `integer`, `date`, ...)
- XML documents *do not* contain any type information
- data can be validated (and types can be added) with additional *languages*

✎ When is a document well-formed, when is it valid?

Definition and Validation Languages

- DTD: Document Type Definition; part of the XML specification
- XML Schema: type and structure definition; expressed in XML itself
- RelaxNG: specified by OASIS (not W3C); simpler than XML Schema
- Schematron: ISO standard; rule-based validation

DTD: Document Type Definition

- set of markup declarations for defining *document types*
- describes *elements*, *attributes*, *entities* and *notations*
- can be *embedded* in an XML document, or as external reference

Examples

- specify document type (DITA map):

```
<!DOCTYPE map PUBLIC "-//OASIS//DTD DITA Map//EN" "map.dtd">
<map id="map" xml:lang="en"/>
```

- embedded document type definitions (HTML):

```
<!DOCTYPE html [ <!ELEMENT html (#PCDATA)*> ]>
<html/>
```

- validation via BaseX: `validate:dtd-report('doc.xml')`

DTD: Document Type Definition

Element types

- introduced by ELEMENT
- ANY: any contents
- EMPTY: no contents
- element content:
  - name, sequence (…,…), choice (…|…)
  - optional quantifier: +, *, ?
- mixed content:
  - #PCDATA: textual content
  - choice, followed by *

Example

```
<!DOCTYPE class [
  <!ELEMENT class   (student)*>
  <!ELEMENT student (name|age)*>
  <!ELEMENT name    (#PCDATA)>
  <!ELEMENT age     (#PCDATA)>
]>
<class>
  <student>
    <name>Jonas Müller</name>
    <age>25</age>
  </student>
  <student>
    <name>John Miller</name>
  </student>
</class>
```

DTD: Document Type Definition

Attribute types

- introduced by ATTLIST, followed by attribute name
- types:
  - ID: unique identifier
  - CDATA: any textual value
  - (…|…): list of allowed values
- modifiers:
  - #REQUIRED: attribute is mandatory
  - #IMPLIED: attribute is optional
  - '…': defaut value

Example

```
<!DOCTYPE docs [
  <!ELEMENT docs (doc)*>
  <!ELEMENT doc  EMPTY>
  <!ATTLIST doc
    id     ID       #REQUIRED
    read   (yes|no) 'no'
    name   CDATA    #REQUIRED
    author CDATA    #IMPLIED>
]>
<docs>
  <doc id='doc1'
       name='Minimal Moralia'
       author='Theodor W. Adorno'/>
  <doc id='doc2' read='yes'
       name='Les Mots'/>
</docs>
```

DTD: Document Type Definition

Entity declarations

- introduced by ENTITY
- similar to *macros*
- *predefined* entities:
  `& < > &quot; %apos;`
- *name* followed by *value*
- value may contain *other entities*
- no *circular definitions*

Example

```
<!DOCTYPE doc [
  <!ELEMENT doc (P)*>
  <!ELEMENT p ANY>
  <!ENTITY arrow "→">
  <!ENTITY ampersand "&amp;">
  <!ENTITY and "&ampersand;">
]>

<doc>
  <p>1 &arrow; one</p>
  <p>1 &and; one</p>
</doc>
```

Restrictions of DTDs

- no *XML syntax*, no *namespace support*, no *data types*, ...

XML Schema

- XML syntax for defining document types
- comes with many pre-defined *data types* (known from XQuery)
- support for *namespaces* and *element hierarchies*
- types can be assigned to XML structure: *schema-aware processing*

✎ Which documents are successfully validated by the document descriptions?

Example: DTD vs. XML Schema

```
<!DOCTYPE docs [
  <!ELEMENT docs (doc)*>
  <!ELEMENT doc EMPTY> ]>

<!-- XML Schema: XML syntax -->
<xsd:schema xmlns:xsd=
  "http://www.w3.org/2001/XMLSchema">
  <xsd:element name="docs">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="doc"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

XML Schema: Example

```
<schema xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="address">
    <complexType>
      <sequence>
        <element name="name"/>
        <element name="born" type="integer"/>
        <element name="country" minOccurs="0" maxOccurs="unbounded">
          <simpleType>
            <restriction base="language">
              <enumeration value="at"/>
              <enumeration value="de"/>
            </restriction>
          </simpleType>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>
```

XML document

```
<address>
  <name>Sigmund Freud</name>
  <born>1856</born>
  <!--optional due to minOccurs="0"-->
  <country>at</country>
</address>
```

RelaxNG

- XML Schema was heavily criticized for its complexity
- ☞ RelaxNG solves many of the deficiencies:
  - *easy* to learn and understand
  - few cases that seem *ambiguous*
  - yet as *powerful* as XML Schema
- *two notations*: XML and compact syntax
- rules can be separated and defined as *named patterns*

Example: DTD vs. RelaxNG

```
<!DOCTYPE docs [
  <!ELEMENT docs (doc)*>
  <!ELEMENT doc EMPTY> ]>

<!-- RelaxNG: XML syntax -->
<rng:element name="docs" xmlns:rng=
  "http://relaxng.org/ns/structure/1.0">
  <rng:element name="doc">
    <rng:empty/>
  </rng:element>
</rng:element>

# RelaxNG: compact syntax
element docs {
  element doc { empty }
}
```

RelaxNG: Example

```
<element name="address"
         xmlns="http://relaxng.org/ns/structure/1.0"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <element name="name"/>
  <element name="born">
    <data type="integer"/>
  </element>
  <zeroOrMore>
    <element name="country">
      <choice>
        <value type="language">at</value>
        <value type="language">de</value>
      </choice>
    </element>
  </zeroOrMore>
</element>
```

XML document

```
<address>
  <name>Thomas Mann</name>
  <born>1875</born>
  <!--optional due to zeroOrMore-->
  <country>de</country>
</address>
```

RelaxNG: Example

```
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
  <start>
    <element name="composers">
      <zeroOrMore>
        <ref name="composer"/>
      </zeroOrMore>
    </element>
  </start>
  <define name="composer">
    <element name="composer">
      <interleave>
        <element name="name"><text/></element>
        <optional>
          <element name="city"><text/></element>
        </optional>
      </interleave>
    </element>
  </define>
</grammar>
```

✎ Does the following document match the schema?

```
<composers>
  <composer>
    <city>Wien</city>
    <name>Anton Webern</name>
  </composer>
  <composer>
    <name>Luigi Nono</name>
  </composer>
</composers>
```

Schematron

- not all rules can be expressed in *XML grammars*
- for example, *data-centric relationships* in documents cannot be expressed with syntax rules
- with Schematron, arbitrary *XPath expressions* can be supplied
- *error messages* are also be supplied by the developer, not the system

☞ easy to use...

Example

```
<schema xmlns=
  "http://www.ascc.net/xml/schematron">
  <pattern name="Check structure">
    <rule context="name">
      <assert test="@id">
         Element 'name' has no id</assert>
      <assert test="count(*) = 2 and
        exists(first) and exists(last)">
        Expected children of 'name':
        'first' and 'last'.</assert>
      <assert test="not(first = last)">
        First and last name must be
        different.</assert>
    </rule>
  </pattern>
</schema>
```