

XML in the Development of Component Systems

Other Schema Languages

Problems of XML Schema

- According to Schematron docs:
 - No support for entities
 - idiomatic or localized data types (date, time) not supported
 - limited support for element content models
 - e.g. start with one fixed element, then allow arbitrary order
 - limited support for idiomatic attribute usage
 - e.g. using attributes to contain or select element content
 - no notion of “document type”
 - in particular, no way of specifying root element
 - complexity
 - implementability
 - “persistent problems with buggy implementations, outside the conservative use of features”

Relax NG

- Schema language for XML
 - Designed by James Clark
- Design goals:
 - simple
 - easy to learn
 - both with XML syntax, and compact non-XML syntax
 - support for namespaces
 - treat attributes uniformly with elements as far as possible
 - unrestricted support for mixed content
 - solid theoretical basis
 - can partner with separate data type language (e.g. XML Schema datatypes)

Relax NG: Resources

- Specification
 - OASIS Relax NG (<http://www.relaxng.org/spec-20011203.html>)
 - ISO/IEC FDIS 19757-2
 - Relax NG compact syntax (<http://www.relaxng.org/compact-20021121.html>)
- Tutorials (<http://www.relaxng.org/>)
- Tools:
 - Jing: Java Relax NG validator, based on SAX2 (James Clark)
 - libxml2
 - conversion tools
 - schema compilers
 - ...

Relax NG: An Example

```
<addressBook>
  <card>
    <name>John Smith</name>
    <email>js@example.com</email>
  </card>
  <card>
    <name>Fred Bloggs</name>
    <email>fb@example.net</email>
  </card>
</addressBook>
```

Relax NG: An Example (2)

```
<element name="addressBook" xmlns="http://relaxng.org/ns/structure/1.0">  
  <zeroOrMore> <!-- alternatively use oneOrMore -->  
    <element name="card">  
      <element name="name">  
        <text/>  
      </element>  
      <element name="email">  
        <text/>  
      </element>  
    </element>  
  </zeroOrMore>  
</element>
```

Relax NG: An Example (3)

```
<element name="addressBook" xmlns="http://relaxng.org/ns/structure/1.0">
  <zeroOrMore >
    <element name="card">
      <element name="name">
        <text/>
      </element>
      <element name="email">
        <text/>
      </element>
      <optional>
        <element name="note">
          <text/>
        </element>
      </optional>
    </element>
  </zeroOrMore >
</element>
```

Relax NG: Alternatives

```
<choice>  
  <element name="name">  
    <text/>  
  </element>  
  <group>  
    <element name="givenName">  
      <text/>  
    </element>  
    <element name="familyName">  
      <text/>  
    </element>  
  </group>  
</choice>
```


Relax NG: Attributes

```
<element name="addressBook">
  <zeroOrMore>
    <element name="card">
      <attribute name="name">
        <text/>
      </attribute>
      <attribute name="email"> <!-- could wrap attribute inside <optional> -->
        <text/>
      </attribute>
    </element>
  </zeroOrMore>
</element>
```

Relax NG: Content models

- Can combine attributes and elements uniformly in choice and group:
 - allow models where properties are given either as nested elements, or as attributes
- Content model of attributes is `<text/>` by default

Relax NG: Pattern Grammars

- Content model can be thought of as a grammar
- Grammar productions specified in XML

Relax NG: Grammar Example

```
<grammar>
  <start>
    <element name="addressBook">
      <zeroOrMore>
        <element name="card"><ref name="cardContent"/></element>
      </zeroOrMore>
    </element>
  </start>

  <define name="cardContent">
    <element name="name"><text/></element>
    <element name="email"><text/></element>
  </define>
</grammar>
```

Relax NG: Data Types

- Support for external data types
 - Set of external data types implementation-defined

```
<element name="number">
```

```
  <data type="integer" datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"/>
```

```
</element>
```

- datatypeLibrary= inherited to nested elements

Relax NG: Enumerations

```
<element name="card">  
  <attribute name="name"/>  
  <attribute name="email"/>  
  <attribute name="preferredFormat">  
    <choice>  
      <value>html</value>  
      <value>text</value>  
    </choice>  
  </attribute>  
</element>
```

- Enumerators are either of type string or type token (built-in)

Relax NG: Lists

```
<element name="vector">  
  <list>  
    <data type="float"/>  
    <data type="float"/>  
  </list>  
</element>
```

- Values are space-separated sequences of elements
- Use oneOrMore for repeating list elements

Relax NG: Interleaving

- Interleaving allows elements to appear in arbitrary order
 - Content models of child elements can arbitrarily interleave
 - `<mixed>p</mixed>` is short for `<interleave><text/>p</interleave>`

```
<element name="addressBook">  
  <zeroOrMore>  
    <element name="card">  
      <interleave>  
        <element name="name"><text/></element>  
        <element name="email"><text/></element>  
      </interleave>  
    </element>  
  </zeroOrMore>  
</element>
```


Relax NG: Modularity

- Can refer to external “sub-grammar” through `<externalRef href=“URI”/>`
 - start production defines included content

...

```
<optional>  
  <element name="note">  
    <externalRef href="inline.rng"/>  
  </element>  
</optional>
```

...

- Additionally, definitions can be included using `<include href=“URI”/>`
 - content of `<include>` can replace definitions of the grammar

Relax NG: Namespaces

- ns= attribute of element defines the namespace
 - can be inherited to nested elements
- ns= also allowed for attributes
 - not automatically inherited
- Using QName for name= in <element> introduces elements in a namespace
 - prefix must have been declared

Relax NG: Name Classes

- Can use wildcard names
 - `<anyName/>` specifies that the element/attribute can have an arbitrary name

```
<define name="anyElement">  
  <element><anyName/>  
    <zeroOrMore>  
      <choice>  
        <attribute><anyName/></attribute>  
        <text/>  
        <ref name="anyElement"/>  
      </choice>  
    </zeroOrMore>  
  </element>  
</define>
```

Relax NG: Annotations

- elements from a namespace other than Relax NG are ignored
- `<div>` element allows to group definitions, e.g. to add common documentation

Relax NG: Combining Definitions

- Multiple <define> elements with same name are allowed
- combine="choice" | "interleave" specifies method of combination
 - For elements, "choice" is typically used
 - For attributes, "interleave" is typically used

Relax NG: Compact Syntax

- Use “traditional” punctuation instead of markup for syntactic structure

```
element addressBook {  
    element card {  
        attribute name { text },  
        attribute email { text }  
    }*  
}
```

- Use EBNF-like syntax for grammars

```
start = AddressBook  
AddressBook = element addressBook { Card* }  
Card = element card { Name, Email }  
Name = element name { text }  
Email = element email { text }
```

Schematron

- Add-on language to specify constraints on documents
 - can be combined with other schema languages (e.g. DTD, XML Schema) to overcome their limitations
- Specification: <http://www.ascc.net/xml/resource/schematron/Schematron2000.html>
- Implementations:
 - Can be implemented trivially in XSLT
 - Stage 1: Transform Schematron Schema into XSLT stylesheet using XSLT
 - Stage 2: Validate document through generated stylesheet
 - Explicit support in several tools:
 - Jing (James Clark)
 - 4Suite (FourThought)
 - Schematron.NET (Daniel Cazzulino)
 - ...

Schematron: Functional Principle

- Idea: An XSLT stylesheet performs the validation
 - No output if valid
 - Output error message if invalid
- Example: a *shortStory* may have an *author* unless it is part of an *anthology*

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
  <xsl:template match='shortStory'>
```

```
    <xsl:if test='../anthology and @author'>
```

```
      Invalid XML
```

```
    </xsl:if>
```

```
  </xsl:template>
```

```
</xsl:stylesheet>
```


Schematron: An Example

```
<rule context="title|description|link">
```

```
  <assert test="parent::channel or parent::image  
    or parent::item or parent::textinput">
```

A <name/> element can only be contained with a channel, image, item or textinput element.

```
</assert>
```

```
<report test="child::*">
```

A <name/> element cannot contain sub-elements, remove any additional markup

```
</report>
```

```
</rule>
```

Schematron: Schema Syntax

- Top-Level element is *schema*
 - xmlns:sch="http://www.ascc.net/xml/schematron"
- *rule* element perform validation
 - context= attribute specifies how to apply rules
 - can contain either *assert* or *report* elements
 - test= attribute specifies what to test
- *pattern* element can be used to group rules
 - each element is checked by atmost one rule per pattern
 - first matching rule is used
- *phase* element can specify several operational modes of a schema
 - *active* child element specifies which patterns to use in a phase