Abstract

• In this tutorial you will learn how to use the RELAX NG schema language, an alternative schema language for XML.
• RELAX NG allows easy and intuitive descriptions of just what is and what is not allowed in an XML document.
• It is simple enough to learn in a few hours, and rich and flexible enough to support the design and validation of every kind of document from the very simple to the very complex.
• Once RELAX NG’s concepts have crossed the blood-brain barrier, you will never be able to take any other schema language very seriously again.

Roadmap

• RELAX NG goals (21 slides)
• The invoice example (9 slides)
• Patterns (22 slides)
• Namespace Routing Language (4 slides)
• Datatypes (12 slides)
• The XLink example (12 slides)
• Tools (14 slides)
• XSD datatypes (16 slides)
DTDs On Warp Drive

- An evolution/generalization of DTDs
- Shares the same basic paradigm
- Based on experience with SGML, XML
- Adds and subtracts features from DTDs
- DTDs can be automatically converted

Reusable Knowledge

- Experts in designing SGML and XML DTDs will find their skills transfer easily
- Design patterns commonly used in XML DTDs can be reused
- Much more mature than if based on a completely new and different paradigm
- Higher degree of confidence in its design is possible

Easy To Learn And Use

- Allows schemas to be patterned after the structure of the documents they describe
- Allows definitions to be composed from other definitions in a variety of ways
- Treats attributes and elements as uniformly as possible

Namespaces

- DTDs are namespace-blind
- RELAX NG fully supports namespaces for elements and attributes
- Namespace support is purely syntactic, not tied to one schema per namespace
Datatyping

- Supports pluggable simple datatype libraries
- Basic library supports strings and tokens
- Full XML Schema Part 2 datatypes available (including facets)
- New libraries can be readily designed and built as needed.

Two Syntaxes, One Language

- Provides two interconvertible syntaxes:
  - an XML one for processing
  - a compact non-XML one for human authoring
- We will learn the compact syntax
- One example of the XML syntax is provided to assist in learning it

Attributes

- “Elements or attributes?”
  - Reasonable people can differ
  - Attributes are treated as much like elements as possible
- Content models include elements as well as attributes
- Attribute defaulting is not done

Unordered Content

- SGML’s & operator allows unordered content models
  - A & B means ((A, B) | (B, A))
- XML DTDs removed & to reduce implementation complexity
- RELAX NG restores & with improvements
Interoperability

- You can convert a DTD to RELAX NG, preserving modularity
- You can author in RELAX NG and deliver in DTD or W3C XML Schema or both
- RELAX NG allows embedded Schematron rules

Pronunciation

- "Relaxing" is the standard way
- Some people say "relax en gee"

AND NOW TO RELAX!
The Invoice Example

An Invoice in XML

<invoice number="640959-0" date="2002-03-12">
  <soldTo>
    <name>Reuters Health Information</name>
    <address>45 West 36th St. New York NY 10018</address>
  </soldTo>
  <shipTo>
    <name>Reuters Health Information</name>
    <address>45 West 36th St. New York NY 10018</address>
  </shipTo>
  <terms>Net 10 days</terms>
  <item ordered="5" shipped="5" unitPrice="7.812">
    Binder, D-ring, 1.5" including shipping
  </item>
  <item ordered="4" shipped="4" backOrdered="2" unitPrice="3.44">
    Fork, Plastic, Heavy, Medium including shipping
  </item>
</invoice>

The Invoice Schema (1)

```xml
(element invoice {
  attribute number { text },
  attribute date { text },
  element soldTo {
    element name { text },
    element address { text }
  },
  element shipTo {
    element name { text },
    element address { text }
  }
})
```

The Invoice Schema (2)

```xml
(element terms { text },
  element item {
    attribute unitPrice { text },
    attribute ordered { text },
    attribute shipped { text },
    attribute backOrdered { text | ?,
      text }
  })
)```
The XML format

```xml
<element name="invoice">
  <attribute name="number"/>
  <attribute name="date"/>
  <element name="soldTo">
    <element name="name">
      <text/>
    </element>
    <element name="address">
      <text/>
    </element>
  </element>
  <element name="shipTo">
    <element name="name">
      <text/>
    </element>
    <element name="address">
      <text/>
    </element>
  </element>
  <element name="terms">
    <text/>
  </element>
  <zeroOrMore>
    <element name="item">
      <attribute name="ordered"/>
      <attribute name="shipped"/>
      <optional>
        <attribute name="backOrdered"/>
      </optional>
      <attribute name="unitPrice"/>
      <text/>
    </element>
  </zeroOrMore>
</element>
```

Things To Note

- The structure of the schema parallels the structure of the document
- Element content models include attributes as well as child elements
- The optional attribute is marked with ?
- "text" is the equivalent of #PCDATA or CDATA

Things To Note

- Commas separate multiple components of a content model when the components appear in the given order
- Of course, the order of attributes does not matter!
- Consequently, attributes can appear in the schema before, after, or mixed in with child elements

Definition Form

```plaintext
start = element invoice {
  attribute number { text },
  attribute date { text },
  element soldTo { name-addr },
  element shipTo { name-addr },
  element terms { text }
  element item {
    attribute ordered { text },
    attribute shipped { text },
    attribute backOrdered { text |?,
      attribute unitPrice, text
    }
  }
}
name-addr =
  element name { text },
```
Definition Form Notes

• In definition form, there must always be a definition of start
• You refer to a rule using just its name
• The order of the rules does not matter; use whatever order makes sense to you (top-down, bottom-up, alphabetical)
• Rule names are only relevant to the schema, and never appear in the document instance

DTD-Style Definitions

```xml
start = element invoice {
    attribute number { text },
    attribute date { text },
    soldTo, shipTo, terms, item
}
soldTo = element soldTo { name-addr }
shipTo = element shipTo { name-addr }
terms = element terms { text }
item = element item { text }
name-addr = name, address
name = element name { text }
address = element address { text }
```

Patterns

• Patterns are the basic building blocks of RELAX NG schemas and rules
• Some kinds of patterns can contain sub-patterns enclosed in braces ({ ... })
Element Patterns

- The form is: `element name { ... }`
- The content model (child elements and attributes) is contained within the braces
- Content models consist of one or more patterns

Attribute Patterns

- The form is: `attribute name { ... }`
- The content model is contained within the braces
- Content models consist of one or more patterns
- You can’t have child elements or attributes within attributes, of course!

Element Patterns

- So what patterns can be inside elements?
  - The text pattern - equivalent to `#PCDATA`
  - Datatypes (next section)
  - Literal strings in quotes:
    - `attribute country { "US" }` means the `country` attribute must have the value `US`.

Attribute Patterns

- So what patterns can be inside attributes?
  - The text pattern - equivalent to `CDATA`
  - Datatypes (next section)
  - Literal strings in quotes:
    - `element country { "US" }` means the `country` element must have the content `US`. 
The Text Pattern

- Matches any amount of arbitrary text, possibly broken up by child elements
- Equivalent to \#PCDATA in elements or CDATA in attributes
- text*, text?, text+ all mean the same as text

Namespaces

- To declare elements and attributes in namespaces, use QNames in element and attribute patterns
- Namespace prefixes are declared like this:
  namespace foo = "(some URI)"
- Namespace declarations must come first in the schema

Default Namespaces

- You can declare a namespace for unprefixed elements (not attributes) like this:
  default namespace = "(some URI)"
- If you want the default namespace to have a prefix too, use:
  default namespace foo = " (some URI)"

Namespaces

Here's an example:
namespace one = "http://example.com/one"
namespace two = "http://example.com/two"
default namespace = "http://example.com"
element para {
  attribute one:class { text },
  attribute two:class { text },
  element line { text }*
}
Choice

- Two patterns separated by | represent a choice between them; the document can match one pattern or the other, not both
- Arbitrary patterns are allowed in a choice: you can have a choice between attributes, between elements, or even between an element and an attribute!

Choice

- A useful case:
  ```
  element data {
    (element id { text } | attribute id { text }},
    text
  }
  ```
- You cannot mix , and | in one list; use parentheses to disambiguate

Choice

Enumerated values use choice like this:
```
element font {
  attribute size {
    "10" | "12" | "14" | "16"
  }
}
```

Interleave

- Interleave is a cross between choice and sequence
- When patterns are combined with &, they all must appear but it can be in any order (as in SGML) …
- … or even mixed together!
Interleave

- So this schema ...
  
  ```
  element head {
      element meta { empty }* &
      element title { text }
  }
  ```
  
- ... matches a head element that has any number of meta child elements (including zero) and a required title child element mixed in anywhere.

Interleave

- Note: In the case of attributes, sequence and interleave are the same thing, because attributes don't have ordering
- So you can use either , or & according to what is the most convenient

Quantifiers

- You can place an *, ?, or + after any pattern to allow it to be repeated:
  - * means zero or more times
  - ? means zero or one times
  - + means one or more times
- These mean the same as in DTD content models, but can be used after any pattern, not just rule names

Comments

- Ordinary comments begin with #
- Documentation comments begin with ## and are copied (in groups) into the XML syntax as a:documentation elements
- The a: prefix represents the namespace of the DTD Compatibility extension to RELAX NG
Datatypes

• A type is a named set of values
• An datatype provides a standardized, machine-checkable representation of a type

Schema Datatypes

• DTDs have only a few datatypes for attributes and only one datatype for elements
• XML Schema provides a long, but fixed, list of datatypes
• RELAX NG can work with any datatype library, including the XSD (XML Schema Datatypes) library

RELAX NG Datatypes

• Datatype patterns are written using QNames
• This use of QNames can't be confused with QNames for elements or attributes, because those are only recognized after the words element and attribute
• The built-in datatypes string and token don't have prefixes and are recognized by all implementations
Declaring Datatype Libraries

- A prefix is declared like this:
  datatypes lib = "(some URI)"
- Datatype library declarations must come first
- RELAX NG processors recognize a system-dependent list of datatype library URIs

Useful Datatypes

- The xsd prefix is predeclared for XML Schema Datatypes
  - xsd:integer represents an integer of arbitrary length
  - xsd:ID, IDREF, and IDREFS are the same as the corresponding DTD types
  - We'll discuss all the xsd types later.

Typed Values

- "0" and token "0" match a "0" character with possible surrounding whitespace
- string "0" matches a "0" character exactly
- xsd:integer "0" matches "0" or "00" or "000" or "-0" or ...

Mixing Datatypes

- Datatypes can't be mixed with anything in the same content model, as in:
  ```
  element foo { 
    xsd:int, xsd:ID, 
    element bar { empty } 
  } 
  ```
  - Character content that represents a datatype cannot have any siblings
Mixing Datatypes

- Choices involving datatypes are fine:

  element foo { xsd:int | xsd: Name }
  element bar { xsd:int |
    element baz { text } }

Datatype Exceptions

- xsd:nonNegativeInteger and xsd:nonPositiveInteger are existing types
- How do we say "non-zero integer"? xsd:integer - xsd:integer "0"
- We can likewise express a token that is not a name:
  xsd:token - xsd:Name

Parameters

- Parameters restrict the values of datatypes
- Each datatype has specific parameters that are legal with it
- An integer between 0 and 999 inclusive:

  xsd:integer {
    minOccurs = "0"
    maxInclusive = "999"
  }

Lists

- List patterns specify that character content or an attribute value is to be separated by whitespace into tokens
- The pattern list {xsd:integer*} matches a list of zero or more whitespace-separated integers
- This example needs * because list itself does not imply repetition
Lists

- The pattern
  \[ \text{list \{xsd:integer, token\}+} \]
  matches the string
  "32 foo 45 bar 76 baz"

The Jing validator

- Written by James Clark, principal author of RELAX NG
- Java based command-line tool
  - Validates schemas
  - Validates documents against schemas
- Accepts either compact or XML syntax
- Optionally enforces DTD ID/IDREF

The Jing validator

- Also usable as a validation library within a Java program
- Provides JAXP (Sun-standard) interface
- Provides native interface
- Validates against other schema languages:
  - W3C XML Schema
  - Schematron
  - Namespace Routing Language (meta)
The Trang translator

• Another James Clark product
• Translates schemas:
  – Input: XML syntax, compact syntax, DTDs
  – Output: XML syntax, compact syntax, DTDs, W3C XML Schema
• Output schemas may be looser than the input schema (accept a superset of what the input accepts)

XSD DATATYPES

Mini-roadmap

• Types (11 slides)
• Facets (5 slides)

XML Schema Datatypes

• A type is a named set of values
• An XML Schema datatype provides a standardized, machine-checkable representation of a type
• XML Schema types can be grouped:
  – numeric, date, boolean, string, misc.
Numeric Types

- Decimal types
- Floating-point types

Decimal Types

- decimal
  - integer
    - nonPositiveInteger
    - negativeInteger
    - nonNegativeInteger
    - positiveInteger
    - unsigned(Long, Int, Short, Byte)
- long, int, short, byte

Floating-point Types

- Only two floating-point types
  - float
  - double
- IEEE ranges (same as Java, all modern hardware)
Date Types

- duration
- date, time, dateTime
- gYear, gMonth, gDay, gYearMonth, gMonthDay

Date Type Examples

duration  P1D  PT3M  P2M
dateTime  2002-06-17T13:45:00
Date  1776-07-04
Time  17:05:00-05:00
gYear  1984
gMonth  ---12
gDay  ---29
gYearMonth  1917-11
gMonthDay  ---09-11

Boolean Type

- Only two values are legal:
  - true (which can also be written 1)
  - false (which can also be written 0)
String Types

- string
  - normalizedString
- token
  - language
  - NMTOKEN(S)
- Name
  - NCName
    - ID, IDREF(S), ENTITY(IES)

Miscellaneous Types

- Raw octet types
  - hexBinary
  - base64Binary
- anyURI
- QName
- NOTATION

Facets

- Allow the creation of new datatypes by restricting the existing ones in one or more ways
- Called params in RELAX NG
- Facets can be grouped into families applicable to datatype families:
  - length, value, pattern
  - enumeration, whiteSpace

Length Facets

- Applicable to string and miscellaneous types
- length facet gives exact length
- minLength and maxLength facets set limits; either or both may be used
- lengths of hexBinary and base64Binary types are measured in octets, not characters
Value Facets

• Applicable to numeric and date types
• minExclusive and minInclusive specify a lower bound; either but not both may be used
• maxExclusive and maxInclusive specify an upper bound; either but not both may be used

Value Facets

• totalDigits specifies the total number of significant digits in a decimal, integer, (non)PositiveInteger, or (non)NegativeInteger value
• fractionDigits specifies the number of fractional digits in a decimal value

Pattern Facet

• Applicable to any type
• Specifies a regular expression that the data must match
• XML Schema: If multiple pattern facets are present, the data must match at least one of them
• RELAX NG: If multiple pattern facets are present, the data must match all of them

MORE INFORMATION

http://www.relaxng.org
http://www.ccil.org/~cowan/relaxng{.ppt,.sxi,.pdf}