Chapter 6: RELAX NG

Chapter 6 Objectives

- RELAX NG syntaxes
- RELAX NG patterns, which are the building blocks of RELAX NG schemas
- Composing and combining patterns into higher-level components for reuse, as well as full schema grammars.
- The remaining features of RELAX NG, including namespaces, name classes, datatyping, and common design patterns

Why Another Schema?

Key Features

- **Simple and easy to learn**
- **Two syntaxes:**
  - **XML syntax**
  - **Compact non-XML syntax***
- **Pattern-based grammar: Solid theoretical basis**
- **XML namespaces support**
- **XML datatypes and user-defined datatypes support**
- **Treats attributes and elements uniformly**
- **Unrestricted support for unordered content**
- **Unrestricted support for mixed content**
- **Highly composable**

- **http://www.relaxng.org**

XML and Compact Syntaxes

- **Trang** is a Java program that can convert the Compact Syntax to the XML Syntax and back. Trang can also convert RELAX NG schemas into DTDs or XML Schemas.
  - http://thaiopensource.com/relaxng/trang.html

- **Jing** is a RELAX NG validator in Java.
  - http://www.thaiopensource.com/relaxng/jing.html

RELAX NG Patterns

Pattern Name	Pattern
element pattern	element name {pattern}
attribute pattern	attribute name {pattern}
text pattern	text

Let's examine name9.rnc

Elements and Attributes

```
element name {
  element first { text },
  element middle { text },
  element last { text },
  attribute title { text }
}
```

RELAX NG (Non-XML)

OR

```
element name {
  attribute title { text },
  element first { text },
  element middle { text },
  element last { text }
}
```

Cardinality

Cardinality Indicator	Meaning
?	Pattern may or may not appear.
+	Pattern can appear one or more times.
*	Pattern can appear zero or more times.
No indicator (default)	Pattern must occur once and only once.

Connector Patterns and Grouping

Pattern Name	Pattern
sequence pattern	*pattern*, *pattern*
choice pattern	*pattern* | *pattern*
interleave pattern	*pattern* & *pattern*
group pattern	(*pattern*)

Sequences and Choices

For example

```
element date { element year{text}, element month{text}, element day{text} }
```

```
<date>
    <year>1959</year>
    <month>08</month>
    <day>14</day>
</date>
```

element a{text} , element b{text} , element c{text} 

element a{text} | element b{text} | element c{text} 

element a{text} , element b{text} | element c{text} 

Not allowed to mix

Sequences and Choices

More examples

element a { text } , (element b { text } | element c { text })

(element a {text}, (elemnt b{text} | (element c {text} , element d {text})))

(element a{text}, (element b{text} | (element c{text},element d{text} *) ?) +

Not allowed to mix without proper grouping

Interleave

For example

```
<person>                          <person>
  <name>Julie Gaven</name>          <phone>555-1234</phone>
  <phone>555-1234</phone>           <name>Julie Gaven</name>
</person>                         </person>
```

```
element person { element name { text } & element phone { text } }
```


Interleave

For example

```
<root>
    <a/>
    <id>54643</id>
    <b/>
    <c/>
</root>
```

id can appear anywhere
a, b, c should appear in order

```
element root {
        element id { text } &
        (element a { text }, element b { text }, element c { text } )
}
```

Enumerated Values

Pattern Name	Pattern
Enumeration Pattern	*datatype value*

Enumerated Values

For example

```
element name {
  attribute title { "Mr." | "Mrs." | "Ms." | "Miss" | "Sir" | "Rev" | "Dr." }?,
  element first { text },
  element middle { text },
  element last { text }
}
```

Valid

```
<?xml version="1.0"?>
<name title="Mr.">
   <first>Joe</first>
   <middle></middle>
   <last>Hughes</last>
</name>
```

Enumerated Values

For example

```
element name {
  attribute title { "Mr." | "Mrs." | "Ms." | "Miss" | "Sir" | "Rev" | "Dr." }?,
  element first { text },
  element middle { text },
  element last { text }
}
```

Invalid

```
<?xml version="1.0"?>
<name title="">
   <first>Maria</first>
   <middle></middle>
   <last>Knapik</last>
</name>
```


Co-Occurrence Constraints

For example

```
<transportation>
  <vehicle type="Automobile" >
    <make>Ford</make>
  </vehicle>
  <vehicle type="Trolley">
    <fare>2.50</fare>
    <tax>1.00</tax>
  </vehicle>
</transportation>
```

Content based on attribute or element value

→ Flexibility!

Complicated, if not impossible, using XML Schema

```
element transportation {
  element vehicle {
    (attribute type { 'Automobile' }, element make { text } ) |
    (attribute type { 'Trolley' }, element fare { text }, element tax { text } )
  }*
}
```

Mixed Content Pattern

Pattern Name	Pattern
mixed pattern	`mixed` {*pattern*}

`mixed` {*pattern*}

is short for

text & *pattern*

Mixed Content Pattern

Example

```
<description>Jeff is a developer and author for Beginning XML <em>4th
 edition</em>.<br/>Jeff <strong>loves</strong> XML!</description>

element description {
  mixed { (element em { text } | element strong { text } | element br { empty })* }
}
```

Note that the textbook places the asterisk incorrectly as:

```
element description {
  mixed { element em { text } | element strong { text } | element br { empty } }*
}
```

Example

Text mixed with one followed by an optional , followed by zero or more

```
element description {
  mixed { element em { text }, element strong { text }?, element br { empty }* }
}
```

The Empty Pattern

Pattern Name	Pattern
empty pattern	`empty`

The Empty Pattern

For example

```
<br/>

element br { empty }

<knows contacts="David_Hunter Danny_Ayers"/>

element knows { attribute contacts { text },  empty }
            or
element knows { empty, attribute contacts { text } }
```

Let's examine contacts14.xml and contacts14.rnc

Combining and Reusing Patterns and Grammars
Named Patterns

Pattern Name	Pattern
named pattern definition	*Pattern Name = pattern*

Named Patterns

For example

FirstNameDef = element first { text }

locationContents = element address { text } |
(element latitude { text }, element longitude { text })

Recursive patterns are allowed. A pattern reference can reference the current pattern name, either directly or indirectly.

element location = { locationContents* }

Named Patterns

Another example

```
start = name
name = element name { nameContents }
nameContents = (
  title?,
  first+,
  middle?,
  last
)
titles = ("Mr." | "Mrs." | "Ms." | "Miss" | "Sir" | "Rev" | "Dr.")
title = attribute title { titles }
first = element first { text }
middle = element middle { text }
last = element last { text }
```

Let's examine contacts15.rnc

Combining Named Pattern Definitions

For example

```
start = name
name = element name {  attribute title { text }? }
name = element name {
  element first { text }+,
  element middle { text }?,
  element last { text }
}
```

Invalid

Combining Named Pattern Definitions

Two combinations possible

Pattern Name	Pattern
named pattern choice	*Pattern Name |= pattern*
named pattern interleave	*Pattern Name &= pattern*

Combining Named Pattern Definitions

For example

```
start = name
name &= element name {  attribute title { text }? }
name &= element name {
  element first { text }+,
  element middle { text }?,
  element last { text }
}
```

Combining Named Pattern Definitions

For example

```
start = name
name |= element name {  attribute title { text }? }
name &= element name {
  element first { text }+,
  element middle { text }?,
  element last { text }
}
```

Invalid

```
start = name
name = element name {  attribute title { text }? }
name &= element name {
  element first { text }+,
  element middle { text }?,
  element last { text }
}
```

Valid

Schema Modularization Using the Include Directive

For example, name10.rnc

```
name = element name { nameContents }
nameContents = (
  title?,
  first+,
  middle?,
  last
)
titles = ("Mr." | "Mrs." | "Ms." | "Miss" | "Sir" | "Rev" | "Dr.")
title = attribute title { titles }
first = element first { text }
middle = element middle { text }
last = element last { text }
```

In another rnc

```
include “name10.rnc”
```

Redefining Included Name Patterns

For example, name10.rnc

include “name10.rnc”

```
include "name10.rnc" {
  nameContents = (
    title?,
    element given { text }+,
    element family { text }
  )
}
```


Removing Patterns with the notAllowed Pattern

For example, in *name* vocabulary

```
start = name
name = element name { nameContents }
nameContents = (
  title?,
  first+,
  middle?,
  last
)
titles = ("Mr." | "Mrs." | "Ms." | "Miss" | "Sir" | "Rev" | "Dr.")
title = attribute title { titles }
first = element first { text }
middle = element middle { text }
last = element last { text }
```

And in *contacts* vocabulary,

```
start = contacts
```

Removing Patterns with the notAllowed Pattern

Solution in *contacts* vocabulary

```
include “name10.rnc” {
  start = notAllowed
}

start |= contacts
```


Extensions and Restrictions

For example

Extend nameContents

```
include "name.rnc" {
  name = element name { nameContentsExt }
}
nameContentsExt = (nameContents, generation?)
generation = element generation { text }
```

Include only maleTitles

```
include "name.rnc" {
 title = attribute title { maleTitles }
}
maleTitles = titles - ("Mrs." | "Ms." | "Miss")
```

-: except

Nested Grammars

Reuse names with various vocabularies by creating a separate scope

For example

Named pattern title for contact's newly added attribute → conflicts with name10.rnc

```
version = attribute version { "1.0" }
source = attribute source { text }
title = attribute title { text }
contacts = element contacts {
  version,
  source?,
  title?,
  contact*
}
```

Nested,

```
name = grammar {
  include "name10.rnc"
}
```


Nested Grammars

Example: not necessarily using include

```
name = grammar {
  start = name
  name = element name { nameContents }
  nameContents = (
    title?,
    first+,
    middle?,
    last
  )
  titles = ("Mr." | "Mrs." | "Ms." | "Miss" | "Sir" | "Rev" | "Dr.")
  title = attribute title { titles }
  first = element first { text }
  middle = element middle { text }
  last = element last { text }
}
```

Next page …

Nested Grammars

Example: reference parent grammar

```
name = grammar {
  start = name
  name = element name { nameContents }
  nameContents = (
    title?,
    parent source?,
    first+,
    middle?,
    last
  )
  titles = ("Mr." | "Mrs." | "Ms." | "Miss" | "Sir" | "Rev" | "Dr.")
  title = attribute title { titles }
  first = element first { text }
  middle = element middle { text }
  last = element last { text }
}
```


Additional RELAX NG Features

Namespaces

default namespace = http://www.example.com/contacts
namespace xhtml = http://www.w3.org/1999/xhtml

```
description = element description {
  mixed { descriptionContents* }
}
descriptionContents = ( em | strong | br )
em = element xhtml:em { text }
strong = element xhtml:strong { text }
br = element xhtml:br { empty }
```

Note the error in the book!!

Additional RELAX NG Features

Names-Classes

Pattern Name	Pattern
element pattern	`element` *name* {*pattern*}
attribute pattern	`attribute` *name* {*pattern*}

- Basic names (includes namespaces)
- Name-class choices and groups
- Namespaces with wildcard
- AnyName

Basic Names (Including Namespaces)

For example

```
element first { text }

element xhtml:em { text }
```

Name-Class Choices and Groups

For example

```
first = element (first | given) { text }
```

Use

```
<name>
  <first>Tom</first>
  <last>Gaven</last>
</name>
```

or:

```
<name>
  <given>Tom</given>
  <last>Gaven</last>
</name>
```


Namespaces with Wildcards

For example

```
description = element description {
  mixed { anyXHTML }*
}
anyXHTML = element xhtml:* { text }
```

Use

```
anyXHTML = element xhtml:* - ( xhtml:script | xhtml:object ) { text }

anyXHTMLorSVG = element (xhtml:* | svg:*) { text }
```

Using AnyName

For example

```
description = element description {
  mixed { anyElementWithText }*
}
anyElementWithText = element * { text }
```

Use

```
anyElement  =  element * { anyElement | text }*
```

Can validate any XML document without attribute!

```
anyElement  =  element * { anyAttribute | anyElement | text }*
anyAttribute = attribute * { text }
```

```
any  =  element * { attribute * {text} | any | text }*
```

Can validate any XML document!

any, anyElement, and anyAttribute:
- NOT RELAX NG keywords
- Use any identifier you wish

Datatypes

Can use XML Schema data types using xsd prefix

For example

```
start = number
number = element number { xsd:integer }
```

Valid

```
<number>1234</number>
```

Invalid

```
<number>John Fitzgerald Johansen Doe</number>
```

Datatypes

XML Schema Facets Example

```
phone = element phone { phoneContents }
phoneContents = (
  kind?,
  PhonePattern
)
PhonePattern = (UsPhonePattern | IntlPhonePattern)
UsPhonePattern = xsd:string { pattern="\d{3}-\d{3}-\d{3}-\d{4}" }
IntlPhonePattern = xsd:string { pattern="\+\d{2}-\d{4}-\d{2}-\d{2}-\d{2}-" }
kinds = ("Home" | "Work" | "Cell" | "Fax")
kind = attribute kind { kinds }
```


List Patterns

Validate a list of whitespace-separated tokens

For example

```
tagNames = (
  "author" |
  "xml" |
  "poetry" |
  "consultant" |
  "CGI" |
  "semantics" |
  "animals"
)
tagList = list { tagNames* }
tags = attribute tags { tagList }
```

Note the errors in the book!!

Use

```
<contact person="Jeff_Rafter" tags="author xml poetry">
```

Comments

- **Start with # and continue to en of line**
- **Not necessarily start from the beginning of line**
- **Adding comments always a good practice**

For example

```
# Defining valid tags
tagNames = (
  "author" |
  "xml" |
  "poetry" |
  "consultant" |
  "CGI" |
  "semantics" |
  "animals"
)
# Note the errors in the book!!
tagList = list { tagNames* }
tags = attribute tags { tagList }
```


Useful Resources

Main Specifications: http://www.relaxng.org/
Validating Parsers/Processors:

- Jing: http://www.thaiopensource.com/relaxng/jing.html
- Trang: http://thaiopensource.com/relaxng/trang.html
- MSV: http://wwws.sun.com/software/xml/developers/multischema/
- Topologi: http://www.topologi.com
- RNV: http://www.davidashen.net/rnv.html

Editors:

- XML Copy Editor: http://xml-copy-editor.sourceforge.net/
- Topologi: http://www.topologi.com/products/tme/index.html
- Oxygen: http://www.oxygenxml.com/
- Nxml mode for GNU Emacs: http://www.thaiopensource.com/download/
- Codeplot Online Collaborative Editor: http://codeplot.com