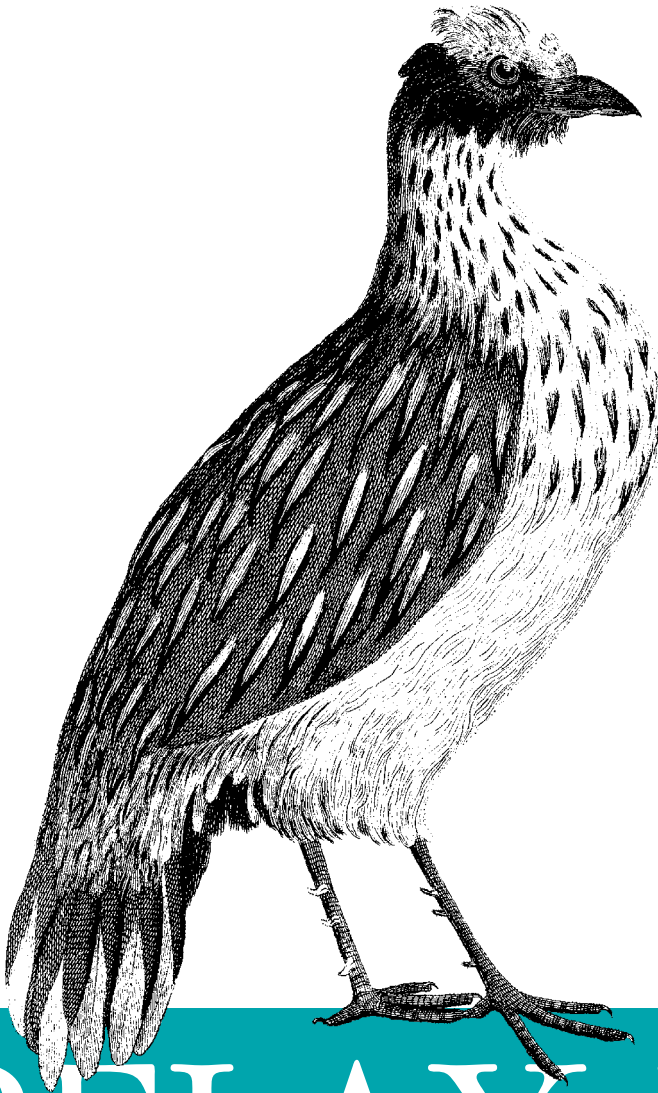


A Simpler Schema Language for XML



RELAX NG

O'REILLY®

Eric van der Vlist

More Complex Patterns

So far, I've described only sequentially ordered groups of elements and text nodes. Now you'll see another class of patterns that describe unordered sequences and choices. Although this class of patterns has no special name in the RELAX NG specification, I refer to them as *compositors* in this book in analogy to the compositors that are defined by W3C XML Schema. The W3C's use of the term compositors matches our usage: the name describes patterns composed of less complex patterns, including the basic patterns we've been looking at such as `element`, `attribute`, and `text`.

One of the key differentiations between compositors and simple patterns is that compositors are patterns that don't directly map to any individual element within the schema. I emphasize this distinction because it can be easy to forget when focusing on a schema instead of the instance document.

The group Pattern

Here is the definition of our character element:

```
<element name="character">
  <attribute name="id"/>
  <element name="name">
    <text/>
  </element>
  <element name="born">
    <text/>
  </element>
  <element name="qualification">
    <text/>
  </element>
</element>
```

In this snippet of the schema, I haven't specified how the different nodes constituting the character element must be composed. RELAX NG recognizes that we have used a group compositor. This group compositor is implied in the XML syntax. You

can see, in the compact syntax, that it looks like a grouping of words: each component of the compositor is separated by a comma:

```
element character {
  attribute id {text},
  element name {text},
  element born {text},
  element qualification {text}}
```

When using the XML syntax, the group compositor may also be explicitly specified, rather than implied. The previous definition is strictly equivalent to this one:

```
<element name="character">
  <group>
    <attribute name="id"/>
    <element name="name">
      <text/>
    </element>
    <element name="born">
      <text/>
    </element>
    <element name="qualification">
      <text/>
    </element>
  </group>
</element>
```

Because the order of attributes isn't considered significant by the XML 1.0 specification, the meaning of the group compositor is slightly less straightforward than it appears at first. Here's the semantic quirk: the group compositor says, "Check that the patterns included in this compositor appear in the specified order, except for attributes, which are allowed to appear in any order in the start tag."

A last thing to keep in mind about group compositors is that, as with compositors in general, there is no such thing as already grouped elements for the pattern to map to in an instance document. The notion of group is specific to the pattern that belongs only in our schema. (This means that there is no hard border in instance documents to isolate nodes inside a group from nodes outside of the group. You'll see later in the chapter that in certain conditions, nodes matching patterns defined outside of a group can be "inserted" in the group.)

The Interleave Pattern

The second compositor examined here, *interleave*, describes a set of unordered patterns—a set of patterns considered valid when they match the content of the instance documents in any order.



As far as validation is concerned, this behavior is similar to the validation of attributes in a “group” compositor up to the point that the algorithms to validate attributes within groups are the same as the algorithm to validate any node in `interleave` compositors. Of course, the validation of `interleave` patterns doesn’t mean that the order of elements and text nodes in the instance document aren’t reported to the application, only that they are allowed to appear in any order.

To specify that character elements may accept child elements in any order, you just need to replace our group pattern with an `interleave` pattern:

```
<element name="character">
  <interleave>
    <attribute name="id"/>
    <element name="name">
      <text/>
    </element>
    <element name="born">
      <text/>
    </element>
    <element name="qualification">
      <text/>
    </element>
  </interleave>
</element>
```

In the compact syntax, `interleave` patterns are marked using an ampersand (&) character as a separator instead of a comma, which is the mark of ordered groups:

```
element character {
  attribute id {text}&
  element name {text}&
  element born {text}&
  element qualification {text}}
```

These two equivalent schemas will validate `character` elements when child elements appear in any order:

```
<character id="PP">
  <name>Peppermint Patty</name>
  <born>1966-08-22</born>
  <qualification>bold, brash and tomboyish</qualification>
</character>
<character id="Snoopy">
  <born>1950-10-04</born>
  <qualification>extroverted beagle</qualification>
  <name>Snoopy</name>
</character>
<character id="Schroeder">
  <qualification>brought classical music to the Peanuts strip</qualification>
  <name>Schroeder</name>
  <born>1951-05-30</born>
</character>
```

Although `interleave` looks straightforward at this point, you'll see that it has more complicated behavior and restrictions. In the last sections of this chapter, we'll look at some of the complexities. You can skip them if they look overwhelming right now, but please remember to come back and revisit them, especially if your `interleave` patterns produce unexpected results or error messages!

The choice Pattern

Let's add some flexibility to the `name` element so we can accept:

```
<name>Lucy</name>
```

and:

```
<name>
  <first>Charles</first>
  <middle>M</middle>
  <last>Schulz</last>
</name>
```

and:

```
<name>
  <first>Peppermint</first>
  <last>Patty</last>
</name>
```

To express this flexibility, use a choice pattern that accepts either a text node or a group of three elements (one of which is optional):

```
<element name="name">
  <choice>
    <text/>
    <group>
      <element name="first"><text/></element>
      <optional>
        <element name="middle"><text/></element>
      </optional>
      <element name="last"><text/></element>
    </group>
  </choice>
</element>
```

The compact syntax uses a pipe, or logical “or” character (`|`) to denote choices:

```
element name {
  text|
  element first{text},
  element middle{text}?,
  element last{text}
}
```

Note that you have to use parentheses to mark the boundary of the group pattern.

Pattern Compositions

In the preceding example, we combined a choice pattern with a group pattern. This process can be expanded so that there is virtually no restriction or limit on the way compositors can be combined. As an example, let's say we want our character element to allow either one name element or the three elements first-name, middle-name (optional), and last-name in any order, but require that they appear before the born and qualification elements. To do that, write:

```
<element name="character">
  <attribute name="id"/>
  <choice>
    <element name="name"><text/></element>
    <interleave>
      <element name="first-name"><text/></element>
      <optional>
        <element name="middle-name"><text/></element>
      </optional>
      <element name="last-name"><text/></element>
    </interleave>
  </choice>
  <element name="born"><text/></element>
  <element name="qualification"><text/></element>
</element>
```

or, with the compact syntax:

```
element character {
  attribute id { text },
  (element name { text }
  | (element first-name { text }
    & element middle-name { text }?
    & element last-name { text })),
  element born { text },
  element qualification { text }
}
```

Note that two levels of parentheses have been added. In the compact syntax, operators determine the nature of compositors (group, interleave, or choice). Operators can't be mixed within one set of parentheses or curly brackets, so you need to use these parentheses to explicitly mark where each compositor begins and ends.

These schemas validate any of the following (and varied) character elements:

```
<character id="PP">
  <first-name>Peppermint</first-name>
  <last-name>Patty</last-name>
  <born>1966-08-22</born>
  <qualification>bold, brash and tomboyish</qualification>
</character>

<character id="PP2">
  <last-name>Patty</last-name>
```

```

<first-name>Peppermint</first-name>
<born>1966-08-22</born>
<qualification>bold, brash and tomboyish</qualification>
</character>

<character id="Snoopy">
  <name>Snoopy</name>
  <born>1950-10-04</born>
  <qualification>extroverted beagle</qualification>
</character>

<character id="Snoopy2">
  <first-name>Snoopy</first-name>
  <middle-name>the</middle-name>
  <last-name>Dog</last-name>
  <born>1950-10-04</born>
  <qualification>extroverted beagle</qualification>
</character>

<character id="Snoopy3">
  <middle-name>the</middle-name>
  <last-name>Dog</last-name>
  <first-name>Snoopy</first-name>
  <born>1950-10-04</born>
  <qualification>extroverted beagle</qualification>
</character>

```

The flexibility and freedom with which you can combine patterns and the lack of restrictions associated with these combinations sets RELAX NG apart from other XML schema languages.

Order Variation as a Source of Information

Before moving on to text patterns and mixed content, the `interleave` pattern deserves more attention. As already noted, calling these content models `unordered` is misleading. Although no order is required by the schema, the nodes will be ordered in instance documents. The order in which they appear in the document can be significant to the applications.

Going back to the example of first and last names: any application managing names will need to know which is a first name and which is a last name. With a little additional effort, they can get the information about whether the first name comes before or after the last name in a XML document. The friendliest of these applications might also want to know whether you prefer to be called by your first or last name first. Do you need to add an additional information item to the schema to carry this information when you could just rely on the order of these elements in the instance document?

In other words, defining content using `interleave` patterns can be seen as degrading the usefulness of a schema because it looks like the information about the order in

which the elements were found will be stripped from the document. That isn't a real problem; XML processors will still present all the order information to your application. In fact, a content model defined with `interleave` patterns allows more combinations than a content model that uses `group` patterns. Thus, with its additional combinations, the `interleave` patterns can let document creators provide additional information that would otherwise disappear into a fixed structure.

The one downside to using `interleave` patterns is that the freedom with which they can be used is unfortunately specific to RELAX NG. If you need to insure that it will also be possible to model your vocabulary with a more rigid schema language such as W3C XML Schema, you will often have to restrict the usage of `interleave` patterns in your RELAX NG schemas.

Text and Empty Patterns, Whitespace, and Mixed Content

So far, we have used text patterns only within `group` patterns. It's important to remember, however, that this pattern doesn't mean simply a text node but rather zero or more text nodes. This statement deserves some exploration.

The reason why text patterns accept zero text nodes is linked to the policy adopted by RELAX NG regarding whitespace. Whitespace processing rules are one of the fuzziest areas in XML. RELAX NG has attempted to find the "least surprising" policy that supports the most common usages. You'll see more whitespace processing when we study datatypes, but for now, let's say that RELAX NG doesn't see any distinction between empty strings; no string at all; strings containing only whitespace before or after an element node; and to a lesser extent, a single text child element containing only whitespace.

For instance, in the following snippet:

```
<foo at1="" at2=" ">
  <bar/>
  <bar></bar>
  <bar>
    <baz/>
    <baz/>
  </bar>
  <bar>
</bar>
</foo>
```

RELAX NG treats as insignificant the values of `at1` and `at2`, the content of the first and second `bar` elements, the text between the third `bar` start tag and the first `baz` element, the text between the two `baz` elements, and even the text within the last `bar`

element. RELAX NG's rules state that the content should match either text or empty patterns. Here are two visible consequences for the patterns we've seen so far:

- Because text patterns match any text node, they must match strings that are either empty or that contain only whitespace. Since there is no difference between empty strings and no string, text patterns match zero strings; i.e., they are always optional.
- Because empty patterns match zero strings and because there is no difference between no string and empty strings or strings containing only whitespace, empty patterns also match strings either empty or containing only whitespace.

In other words, the snippet shown here matches both content models in which all the occurrences mentioned are described as text or empty patterns. If you add the rule—already used a lot but not yet explained—that says you don't need to explicitly express empty patterns between elements, the two schemas will both validate this instance document:

```
<element xmlns="http://relaxng.org/ns/structure/1.0" name="foo">
  <attribute name="at1"><text/></attribute>
  <attribute name="at2"><text/></attribute>
  <oneOrMore>
    <element name="bar">
      <choice>
        <text/>
        <oneOrMore>
          <element name="baz"><text/></element>
        </oneOrMore>
      </choice>
    </element>
  </oneOrMore>
</element>
```

or:

```
<element xmlns="http://relaxng.org/ns/structure/1.0" name="foo">
  <attribute name="at1"><empty/></attribute>
  <attribute name="at2"><empty/></attribute>
  <oneOrMore>
    <element name="bar">
      <choice>
        <empty/>
        <oneOrMore>
          <element name="baz"><empty/></element>
        </oneOrMore>
      </choice>
    </element>
  </oneOrMore>
</element>
```

After having seen why text patterns have to be optional, you need to see why it's also useful for them to match multiple instances. When a text pattern is used with a group or choice pattern, it doesn't make any difference because text nodes are

merged when they are contiguous or separated by infoset items not checked by RELAX NG, such as comments or *processing instructions* (PIs). Within a group or a choice, there is no difference between a pattern that matches one or one or more text nodes. The only place it can make a difference is thus within interleave compositors, and that's the reason why this specificity has been introduced. Document-oriented applications, including XHTML, TEI, and DocBook, provide numerous examples of elements that accept text and embedded elements in any order (called *mixed content*), and in this case, it makes no sense to limit the number of text nodes.

To introduce a mixed content model, let's extend the title element to include zero or more links using some a elements with href attributes:

```
<title xml:lang="en">Being a
  <a href="http://dmoz.org/Recreation/Pets/Dogs/">Dog</a>
  Is a Full-Time
  <a href="http://dmoz.org/Business/Employment/Job_Search/">Job</a>
</title>
```

The content of the new title element can be described as an interleave pattern that allows zero or more a elements and zero or more text nodes. The text pattern matches zero or more text nodes, which will allow us to avoid specifying its cardinality. You can just write:

```
<element name="title">
  <interleave>
    <attribute name="xml:lang"/>
    <zeroOrMore>
      <element name="a">
        <attribute name="href"/>
        <text/>
      </element>
    </zeroOrMore>
    <text/>
  </interleave>
</element>
```

or, using the compact syntax:

```
element title {
  attribute xml:lang {text}&
  element a {attribute href {text}, text}*&
  text
}
```

Because this definition is quite verbose for a common task, RELAX NG has introduced a specific mixed compositor, which has the same meaning as “interleave including a text pattern.” These schemas are strictly equivalent to:

```
<element name="title">
  <mixed>
    <attribute name="xml:lang"/>
    <zeroOrMore>
      <element name="a">
```

```

    <attribute name="href"/>
    <text/>
  </element>
</zeroOrMore>
</mixed>
</element>

```

The mixed compositor is marked using a mixed pattern in the compact syntax and can be written as:

```

element title {
  mixed {
    attribute xml:lang {text}&
    element a {attribute href {text}, text} *
  }
}

```

Why Is It Called Interleave?

If interleave were only about defining unordered groups, why would it be called interleave and not `unorderedGroup` or something similar? The interleave pattern has hidden sophistication. It isn't only a definition for unordered groups, it's also a definition for unordered groups that let their child nodes intermix within subgroups. Mixing is allowed even when these groups are ordered groups. I promise this concept is simpler than it looks in this semiformal definition. An example will make it easier to grasp.

That ordered groups can be immersed in an unordered group might be surprising. Let's try a real-world metaphor to illustrate it. Imagine that the elements of a XML document are like a bunch of tourists visiting a museum; you can then define the unordered sets as all the tourists visiting. The ordered groups of tourists, who are within the unordered set, are following guides. There are many ways to immerse ordered groups within the unordered set of museum visitors and to mix ordered groups together. The interleave pattern describes one specific way to effect this immersion: when the museum is an interleave pattern, the ordered groups preserve only the relative order of their members. This not only allows individual tourists to insert themselves within a group, but also lets two groups interleave their members.

To return to XML and RELAX NG, let's examine the following schema:

```

<element xmlns="http://relaxng.org/ns/structure/1.0" name="museum">
  <interleave>
    <element name="individual"><empty/></element>
    <group>
      <element name="group-member1"><empty/></element>
      <element name="group-member2"><empty/></element>
    </group>
  </interleave>
</element>

```

or, using the compact syntax:

```
element museum {
  element individual {empty} &
  (
    element group-member1 {empty},
    element group-member2 {empty}
  )
}
```

An `individual` represents an individual visiting the museum, while elements `group-member1` and `group-member2` represent visitors in a group. Because `interleave` patterns are not ordered groups, the following instance documents are valid:

```
<museum>
<individual/>
<group-member1/>
<group-member2/>
</museum>
```

and:

```
<museum>
<group-member1/>
<group-member2/>
<individual/>
</museum>
```

These documents are instances in which the element `individual`, which matches the first pattern in the `interleave` pattern (i.e., the `element` pattern), is either before or after the elements `group-member1` and `group-member2`, which match the `group` pattern—the second subpattern of the `interleave` pattern. Because the `interleave` pattern allows that the nodes matching its subpattern to be mixed, the schema also validates this third combination:

```
<museum>
<group-member1/>
<individual/>
<group-member2/>
</museum>
```

On the other hand, because of how the elements are ordered in the `group` declaration of the schema, all the combinations in which the relative order between group members aren't respected are invalid. Here's an example of such an invalid combination:

```
<museum>
<group-member2/>
<individual/>
<group-member1/>
</museum>
```

The `interleave` pattern can also be used to mix two groups of patterns. In this case, the relative order of the element of each group is maintained, but the elements of

different groups may appear in any order and the groups may be interleaved. For an example, let's look at the following schema:

```
<element xmlns="http://relaxng.org/ns/structure/1.0" name="museum">
  <interleave>
    <group>
      <element name="group1.member1"><empty/></element>
      <element name="group1.member2"><empty/></element>
    </group>
    <group>
      <element name="group2.member1"><empty/></element>
      <element name="group2.member2"><empty/></element>
    </group>
  </interleave>
</element>
```

or, using the compact syntax:

```
element museum{
  (
    element group1.member1 {empty},
    element group1.member2 {empty}
  ) & (
    element group2.member1 {empty},
    element group2.member2 {empty}
  )
}
```

This schema validates documents such as:

```
<museum>
  <group1.member1/>
  <group1.member2/>
  <group2.member1/>
  <group2.member2/>
</museum>
```

and:

```
<museum>
  <group2.member1/>
  <group2.member2/>
  <group1.member1/>
  <group1.member2/>
</museum>
```

where the groups are kept separated, but also:

```
<museum>
  <group1.member1/>
  <group2.member1/>
  <group2.member2/>
  <group1.member2/>
</museum>
```

or:

```
<museum>
  <group1.member1/>
  <group2.member1/>
  <group1.member2/>
  <group2.member2/>
</museum>
```

in which the groups are interleaved.

Mixed Content Models with Order

You have seen that a pattern interleaved with a group is allowed to appear anywhere between the patterns of the group. This feature may be used with a text pattern to define ordered mixed-content models, in which the text nodes may appear anywhere but the order of the elements is fixed. These content models are quite unusual in XML. A use case might be a data-oriented vocabulary such as our library, in which optional text can be inserted to provide more user-friendly documentation:

```
<character id="Lucy">
  <name>Lucy</name> made her first apparition in a Peanuts strip on
  <born>1952-03-03</born>, and the least we can say about her is that she is
  <qualification>bossy, crabby and selfish</qualification>.
</character>
```

If you want to fix the order of the child elements, just embed a group pattern inside a mixed pattern:

```
<!--This schema is INVALID-->
<element name="character">
  <mixed>
    <attribute name="id"/>
    <group>
      <element name="name">
        <text/>
      </element>
      <element name="born">
        <text/>
      </element>
      <element name="qualification">
        <text/>
      </element>
    </group>
  </mixed>
</element>
```

Per the definition of the mixed pattern, this is equivalent to:

```
#This schema is invalid
<element name="character">
  <interleave>
    <attribute name="id"/>
```

```

<text/>
<group>
  <element name="name">
    <text/>
  </element>
  <element name="born">
    <text/>
  </element>
  <element name="qualification">
    <text/>
  </element>
</group>
</interleave>
</element>

```

The text pattern matches text nodes before, after, or between the elements of the group, but as you’ve seen with the previous museum example, the order of the elements in the group will still be enforced. The compact syntax uses the mixed pattern with commas between subpatterns to express this:

```

element character {
  mixed {
    attribute id {text},
    element name {text},
    element born {text},
    element qualification {text}
  }
}

```

You have already seen that the compact syntax mixed pattern can be employed using ampersands and commas to define unordered and ordered mixed patterns. An “or” (|) can also interleave text nodes in choice patterns:

```

element foo{
  mixed {
    (
      element in1.1 {empty},
      element in1.2 {empty}
    ) | (
      element in2.1 {empty}&
      element in2.2 {empty}
    )
  }
}

```

This mixed pattern is interleaving text nodes into either a group (denoted by a comma) of `in1.1` and `in1.2` elements or (as shown by the pipe character) an interleave pattern (denoted by an ampersand) of elements `in2.1` and `in2.2`. In the first case, because of the semantics of group patterns, the order between elements is fixed, while in the second case, the order doesn’t matter. Mixed-choice contents don’t constitute new content models. They are equivalent to choices of mixed-content models, and so, you can rewrite this schema as:

```

element foo{
  (
    mixed{
      element in1.1 {empty},
      element in1.2 {empty}
    }
  ) | (
    mixed{
      element in2.1 {empty}&
      element in2.2 {empty}
    }
  )
}
}

```

A Restriction Related to interleave

You'll see the restrictions of RELAX NG in Chapter 15, but I need to mention the principal restriction related to the interleave compositor, as it might affect you at some point if you combine mixed-content models.

Let's extend our title element to allow not only links (a elements) but also bold characters marked by a b element:

```

<title xml:lang="en">Being a
  <a href="http://dmoz.org/Recreation/Pets/Dogs/">Dog</a>
  Is a <b>Full-Time</b>
  <a href="http://dmoz.org/Business/Employment/Job_Search/">Job</a>
</title>

```

Because text can appear before the a elements, between a and b, and after the b element, you might be tempted to write the following schemas:

```

<element name="title">
  <interleave>
    <attribute name="xml:lang"/>
    <text/>
    <zeroOrMore>
      <element name="a">
        <attribute name="href"/>
        <text/>
      </element>
    </zeroOrMore>
    <text/>
    <zeroOrMore>
      <element name="b">
        <text/>
      </element>
    </zeroOrMore>
    <text/>
  </interleave>
</element>

```

or:

```
element title {
  attribute xml:lang {text}
  & text
  & element a {attribute href {text}, text} *
  & text
  & element b {text} *
  & text
}
```

Running the Jing validator against this schema raises the following error:

```
Error at URL "file:/home/vdv/xmlschemata-cvs/books/relaxng/examples/RngMorePatterns/
interleave-restriction2.rnc",
line number 1, column number 2: both operands of "interleave" contain "text"
```

This error results because there can be only one text pattern in each interleave pattern. You have seen that text patterns match zero or more text nodes, and in this case, the remedy is simple enough: the schema must be rewritten as:

```
<element name="title">
  <interleave>
    <attribute name="xml:lang"/>
    <text/>
    <zeroOrMore>
      <element name="a">
        <attribute name="href"/>
        <text/>
      </element>
    </zeroOrMore>
    <zeroOrMore>
      <element name="b">
        <text/>
      </element>
    </zeroOrMore>
  </interleave>
</element>
```

or:

```
element title {
  attribute xml:lang {text}
  & text
  & element a {attribute href {text}, text} *
  & element b {text} *
}
```

This new schema is perfectly valid and does what we tried to do with our invalid schema.

In this example, diagnosing the problem was very simple, but in practice, the situation is often more complex. There can be conflicting text patterns belonging to different subpatterns of interleave or mixed patterns. When using pattern libraries (as shown in Chapter 10), the conflicting text patterns often belong to different RELAX

NG grammars, making it still more difficult to pinpoint the problem. To make it even worse, the error messages from the RELAX NG processors are often quite cryptic, in this case telling you there are conflicting text patterns in `interleave` patterns without saying where they come from. Unfortunately, for now at least, you'll have to figure this out by yourself.



The reason behind the restriction of only one text pattern in each `interleave` pattern is to optimize RELAX NG implementations using the derivative method described by James Clark. When processing mixed-content models, instead of processing each text node, these implementations can simply memorize the fact that this is mixed content and ignore each text node. To do so, the implementation needs to be able to quickly find if a content model mixed or not mixed. That's where the restriction makes a difference in terms of programming complexity and execution speed.

A Missing Pattern: Unordered Group

We have seen that the `interleave` pattern associates two different features and is both an unordered group and something that alters the way subgroups can be combined. These two features aren't totally independent because mixing child nodes is meaningful only when the order of the subgroups isn't maintained, but they aren't totally dependent either. In theory, it's possible to define a pattern with a meaning of "unordered group" that doesn't interleave child nodes and keeps groups unaltered.

This pattern doesn't exist in RELAX NG for two reasons. First, it helps keep the language as simple as possible. Also, although it is built on top of an abstract mathematical model, RELAX NG is also built on top of the experience of its authors who have wanted to focus on general usages and best practices amongst the XML community. The lack of a "unordered group with no interleaving" hasn't been reported as a real-world limitation so far.