1020234Origins

1030245Origins

Extending XML with SHORTREFs, specified in RELAX NG

Mario Blazevic

2012-07-31 Tue

Grammar-driven Markup Generation

- Balisage 2010
- Input: partially marked up, well-formed XML
- Output: valid XML, conforming to the target RELAX NG schema
- Implementations in Haskell and in OmniMark

Example input

- Well-formed almost-XHTML

```
<ul>
  Here's a list
  <li>First item</li>
  <li>Some subitems:</li>
  <ul>
     <li>First subitem</li>
     <li>Second subitem</li>
  </ul>
  </ul>
```

Example output

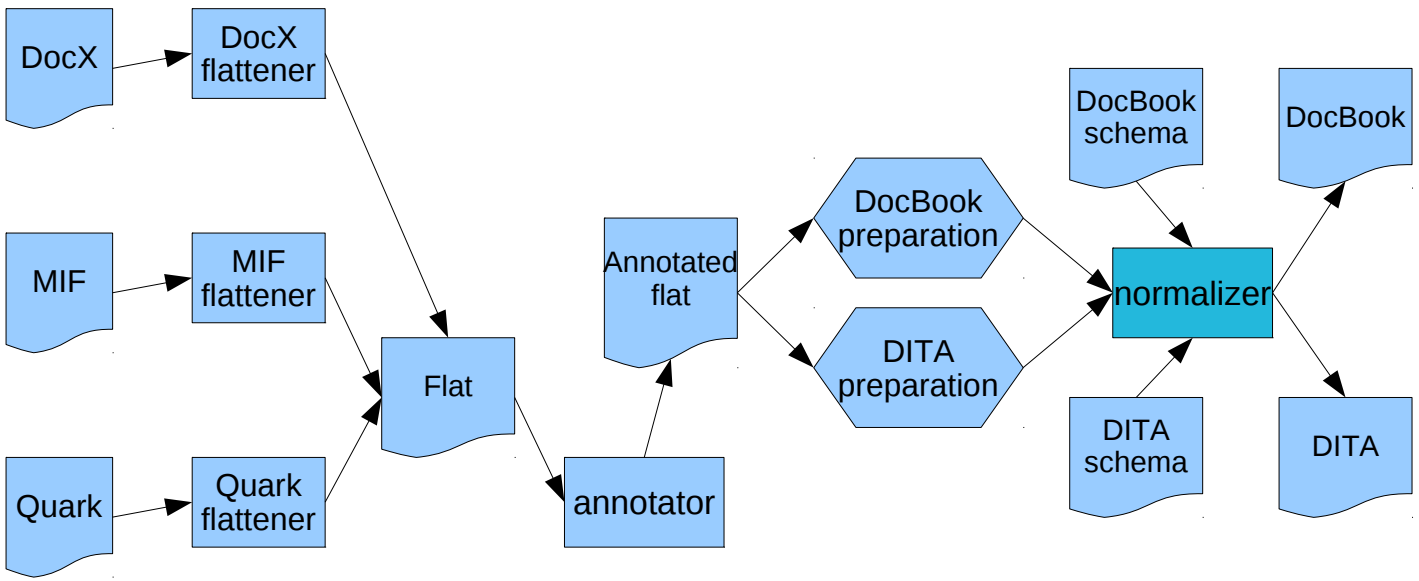
- Valid XHTML 1.0 Strict

```
<ul>
  <li>Here's a list</li>
  <li>First item</li>
  <li>Some subitems:</li>
  <li>
    <ul>
       <li>First subitem</li>
       <li>Second subitem</li>
    </ul>
  </li>
</ul>
```

Limits

- The normalizer can only insert element tags with no attributes.
- The output of the normalizer always conforms to the target schema.
- Parts of input are dropped only if they cannot be wrapped into the target schema.
- **Text nodes are atomic.**
  - this restriction is about to be removed

WYSIWYG to semantically rich XML



Simple idea

- Extract text and presentational markup from a WYSIWYG document.
- Users annotate the WYSIWYG markup with semantics.
- Convert the annotated WYSIWYG markup to target markup.
- ...
- Profit!

The problem: WYSIAYG

- If it ain't italic, bold, or underlined, it ain't marked up.
- It's bold, all right, but what kind of bold?

From the Control Center, select **File** > **Launch** > **Manager**, then follow steps from Figure 12 on page 149.

The customer is always right

- And while you're at it, turn this MS Word input:

From the Control Center, select **File** > **Launch** > **Manager**, then follow steps from Figure 12 on page 149.

- into this DITA output:

```
From the <keyword>Control Center</keyword>, select
<menucascade>
  <uicontrol>File</uicontrol>
  <uicontrol>Launch</uicontrol>
  <uicontrol>Manager</uicontrol>
</menucascade>,
then follow steps from
<xref href="Figure12.dita"/>.
```

The lack of markup in authoring

- No source of documentation is fully marked up.
  - WYSIWYG
  - Mathematical and technical notation
  - User-generated content on the Web
- There are too many details to mark up,
- and a GUI doesn't help.

User-generated content on the Web

- by and large, is not entered in XML
- wikimatrix.org
- XML has a verbosity problem.

Counter-example: XSLT

- That's pure XML, right?
- Wrong:
  - XPath is not XML,
  - neither is the expression sub-language in XSLT.

Verbosity, again

- Nobody likes entering the date **2012-07-31** as

```
<year>2012</year><month>7</month><day>31</day>
```

- nor the XPath **foo//bar@baz** as

```
<relative-location-path>
  <child>foo</child>
  <descendant>bar</descendant>
  <attribute>baz</attribute>
</relative-location-path>
```

- Yet the XML forms are easier to process. What is to be done?

Simple Data Types

- The problem has long been recognized, of course. . .
  - the solution, not so much.
- Generally, the standards just accept certain character strings as special:
  - the strings are validated against their data type,
  - but their parsed structure is not made explicit.

DTD data types

- SGML DTDs allow 14 different predefined data types.
- XML DTDs reduce this number down to 8:

  ```
  CDATA
  ID IDREF IDREFS
  ENTITY ENTITIES
  NMTOKEN NMTOKENS
  ```

  plus the enumerated types.
- Notations are the only other extension mechanism.

XML Schema data types

- There are 19 primitive and 25 derived data types built into W3C XML schema.
- More importantly, users can define their own data types.
- The most powerful mechanism provided are the regular expressions.
  - One could define a date, if it wasn't already built in:

    ```
    [12][0-9]{3}-(0?[1-9]|11|12)-([012]?[0-9]|30|31)
    ```

  - or an abbreviated XPath:

    ```
    /|(/|//)?(\*|\i\c*)((/|//)(\*|\i\c*))+
    ((/|//)@\i\c*
     |(comment|text|processing-instruction)\(\))?
    ```

XML Schema: type dichotomy

- There's the simple date type

```
[12][0-9]{3}-(0?[1-9]|11|12)-([012]?[0-9]|30|31)
```

- and then there's the complex date type:

```
<xs:complexType name="date">
  <xs:sequence>
   <xs:element name="year" type="xs:number"/>
   <xs:element name="month" type="month"/>
   <xs:element ref="day" type="day"/>
  </xs:sequence>
 </xs:complexType>
```

- Choose either the concise or the precise notation.

RELAX NG

- The complex types evolve
  - modular
  - removed XML Schema restrictions
  - orthogonal composition operators
  - strictly more powerful than regular expressions
- The simple types stagnate
  - carried over wholesale from W3C XML Schema

Use the force

- Forget about simple data types
- Use full RELAX NG grammar to specify the strings' structure
- Then, make the element tags omissible

The date example

- There's the simple date type

```
[12][0-9]{3}-(0?[1-9]|11|12)-([012]?[0-9]|30|31)
```

- and then there's the complex date type:

```
date= element date {year, date_marker,
                    month, date_marker, day}
year= element omissible:year {positiveInteger}
month= element omissible:month {
         positiveInteger {maxInclusive=12}}
day= element omissible:day {
       positiveInteger {maxInclusive=31}}
date_marker= element terminal:date_marker{ "-" }
```

Use the force

- Abbreviated XPath, as a simple data type:

```
xsd:string {pattern=
  "/|(/|//)?(\*|\i\c*)((/|//)(\*|\i\c*))+
   ((/|//)@\i\c*
    |(comment|text|processing-instruction)\(\))?"}
```

- Abbreviated XPath, in RNC:

```
XPath= element XPath {absolute|absoluteDescendant|r
absolute= element absolute {child_marker, relative?
absoluteDescendant= element absoluteDescendant {des
relative= element relative {step+}
step= element step {(axisSpecifier, nodeTest, predi
axisSpecifier= element axisSpecifier {(axisName, ax
parent = element parent { parent_marker }
self = element self { self_marker }
```