Analytical XSLT

Liam Quin
Delightful Computing
https://www.fromoldbooks.org/
XML Prague 2020

Outline

- Converting XML to a new version of a DTD
- Or an older, or different, version
- A big DTD
- With lots of elements and attributes

Scenario: A New Revision

- Suppose you're working with a big DTD ...
- Your organization decides to move to a later version of it ...
- Or, a customer to whom you send XML does ...
- What do you do?

Wait? Did he just say DTD?

- Yes, people still use DTDs.
- The techniques in this paper will work for other grammar formats, but currently the tool presented is DTD-only.
- We'll come back to this topic later.

So, You Need to Review:

- What has changed in the DTD
- What would no longer validate with an identity transformation?
- What *might* no longer validate with an identity transform?

First, Look for Changes

- Look for a change log; these are usually either manually edited, helpful, but incomplete, or automatic from commit logs and include sequences like Change X to Y, then, later, change Y to Z
- Look for comments in the DTD
- Then you might try ...

Text Diffs

- You can use the Unix/Linux *diff* command on the DTDs, but this is line-based, not declaration-based.
- You can "flatten" the DTD to expand parameter entities but or-groups in a different order are a common result & give false positives.

XML-Aware Tools

- The ancient and respectable DTD Diff
  - Gives a *diff*-like summary of the changes
  - May require a "flattened" DTD (e.g. for JATS)
  - Better than Unix *diff* for this purpose.

DTD Comparator

- Part of a Java-based DTD-Analyzer package that includes also a DTD flattener tool;
- Produces an HTML report and an initial XSLT stylesheet that you edit;
- I didn't find this before writing Eddie, *but* ...

All these approaches suck !

- They are based on a waterfall model in which you analyse the differences, understand them, then write code to transform documents.
- But DTD, like most requirements, *evolve* continuously.
- And you don't need to handle all changes ...

Meet Eddie 2.

- Currently a (working) prototype;
- Uses a SAX API to read DTDs;
- Generates a much more extensive report;
- Generates a stylesheet that you do *not* edit, but instead *import*;
- Uses a simple configuration file.

Traditional XSLT

- Common cycle:
  - Write a transformation
  - Run it on test documents
  - See the validation errors
  - Scratch your head and puzzle them out.

Analytical XSLT

- Common cycle:
  - Write a transformation
  - Run it on test documents
  - See errors in terms of the *input*
  - Mark off elements you handled
  - Repeat.

How Eddie 2 Helps This

- The generated XSLT uses xsl:message to warn about input elements that may cause problems;
- Mark elements as handled in the conf file, or as safe to copy, and they no longer generate warnings;
- The report shows "unsafe" elements have not yet been marked as handled: a to-do list.

role

role not in configuration file

Children differ

index-term, index-term-range-end, inline-media

(all these children are in the Eddie2 configuration already)

Or-groups with different children

```
<!ELEMENT role "(
        #PCDATA|email|ext-link|uri|inline-supplementary-material|related-article|
        related-object|hr|bold|fixed-case|italic|monospace|overline|overline-start|
        overline-end|roman|sans-serif|sc|strike|underline|underline-start|underline-
        end|ruby|alternatives|inline-graphic|inline-media|private-char|chem-struct|
        inline-formula|tex-math|mml:math|abbrev|index-term|index-term-range-end|
        milestone-end|milestone-start|named-content|styled-content|fn|target|xref|sub|
        sup|x
)*">
```

```
<!ELEMENT role "(
        #PCDATA|email|ext-link|uri|inline-supplementary-material|related-article|
        related-object|hr|bold|fixed-case|italic|monospace|overline|overline-start|
        overline-end|roman|sans-serif|sc|strike|underline|underline-start|underline-
        end|ruby|alternatives|inline-graphic|private-char|chem-struct|inline-formula|
        tex-math|mml:math|abbrev|milestone-end|milestone-start|named-content|
        styled-content|fn|target|xref|sub|sup|x
)*">
```

- resource-id
- resource-name
- resource-wrap
- ✖response
- role
- ✔roman
- ✖rp
- ✖rt
- ✖ruby
- ✔sans-serif
- ✔sc
- ✖season
- ✔sec
- ✖sec-meta
- ✔see
- ✔see-also
- ✔self-uri
- ✔series
- ✔series-text
- ✔series-title
- ✔sig
- ✔sig-block
- ✖size
- ✔source
- ✔speaker
- ✖speech
- ✖state
- ✖statement

Attributes differ:

degree-contribution, vocab, vocab-identifier, vocab-term, vocab-term-identifier

Attributes in source but not destination:

degree-contribution CDATA #IMPLIED 0

vocab CDATA #IMPLIED 0

vocab-identifier CDATA #IMPLIED 0

vocab-term CDATA #IMPLIED 0

vocab-term-identifier CDATA #IMPLIED 0

content-type	CDATA	#IMPLIED
degree-contribution	CDATA	#IMPLIED
id	ID	#IMPLIED
specific-use	CDATA	#IMPLIED
vocab	CDATA	#IMPLIED
vocab-identifier	CDATA	#IMPLIED
vocab-term	CDATA	#IMPLIED
vocab-term-identifier	CDATA	#IMPLIED
xml:base	CDATA	#IMPLIED
xml:lang	NMTOKEN	#IMPLIED
content-type	CDATA	#IMPLIED
id	ID	#IMPLIED
specific-use	CDATA	#IMPLIED
xml:base	CDATA	#IMPLIED
xml:lang	NMTOKEN	#IMPLIED

Element role is found in:

collab, contrib, contrib-group, element-citation, mixed-citation, person-group, product, related-article, related-object, sig-block

[same in both source and destination]

Report Demo

- Show content models and hover goodness
- Show index with TODO/conf info
- Show content model with hover-dots
- Ask for suggestions

The Generated XSLT

- Eddie 2 makes a template for each element in the source DTD, with comments comparable to the report.
- The template also produces messages:

Messages

- Element sock contains blister not in source DTD
- Element sock has price attribute not in source DTD
- Element sock has fabric attribute with incompatible value list
- Element sock has xmlns:xlink attribute declared as CDATA in source but #FIXED in destination

Message Methodology

- Messages from Eddie 2 don't contain filenames or line numbers, so you can sort them and count how many there are of each (*sort | uniq -c*).
- Use this to address the most common ones first.
- The messages are in terms of the input DTD, not the input files or output XML.

In Production

```
<xsl:import href="lib/eddie2.xsl" use-
when="$use-eddie2 = 'yes' " />
```

- To do this, declare *use-eddie2* as a static parameter:

```
<xsl:param name="use-eddie2" static="yes"
select=" 'n' " />
```

Does it Work?

- A conversion from one DTD to another, previously unseen, without documentation (but with test input files) can be done in a few hours, with repeated runs of E*ddie* 2 and addressing the most common messages.
- The report provides a useful level of documentation.
- But it doesn't make it trivial: ...

When it doesn't help

- If your input is BITS and your output is HTML (say), the only use is the report to see content models and make a to-do list.
- Doesn't know about DTD-specific items, such as JATS date representation.
- So...

Future Work

- DTD-specific plugins;
- Turn from working prototype to production code;
- Some XSLT coverage reporting;
- Element renaming;
- Parameter Entities in the report?
- Support RNG and XSD? *Hard*!

Questions

Liam Quin

Delightful Computing.com

liam@fromoldbooks.org