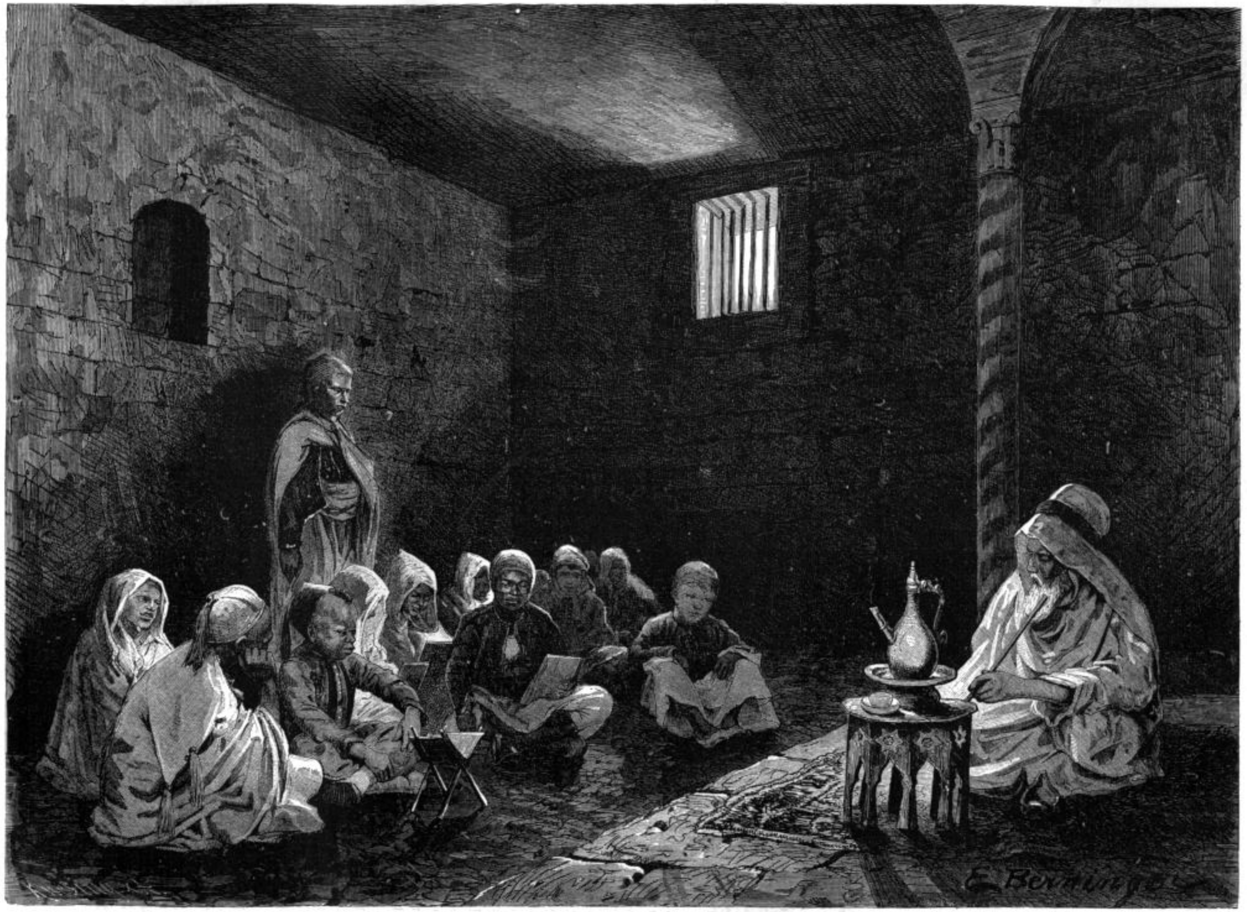
Parſing DTD Files in XSLT to *Expoſe* the Declarations they Contain.

Liam Quin
Delightful Computing

Balisage 2024

Consulting & Teaching



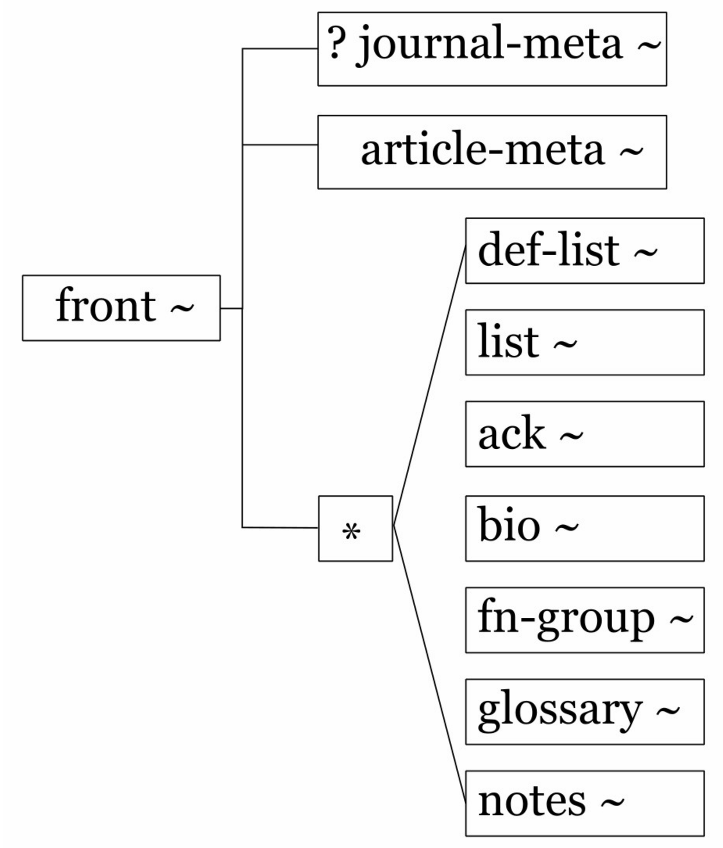
Beginning a Journey

The *author* needed to process element declarations, including their attributes and content models, in XSLT.

So it set out on a....

Journey to Fairyland

Why? Diagramming



Why? Tooling & Exploring

- For writing transformations to convert documents from one DTD to another;
- For exploring:
  - Which elements in the DTD have no matching template?
  - Which elements in the DTD do not occur in sample data?
  - Which xmlns:* attributes have FIXED values?

Gratuitous Tree Diagram

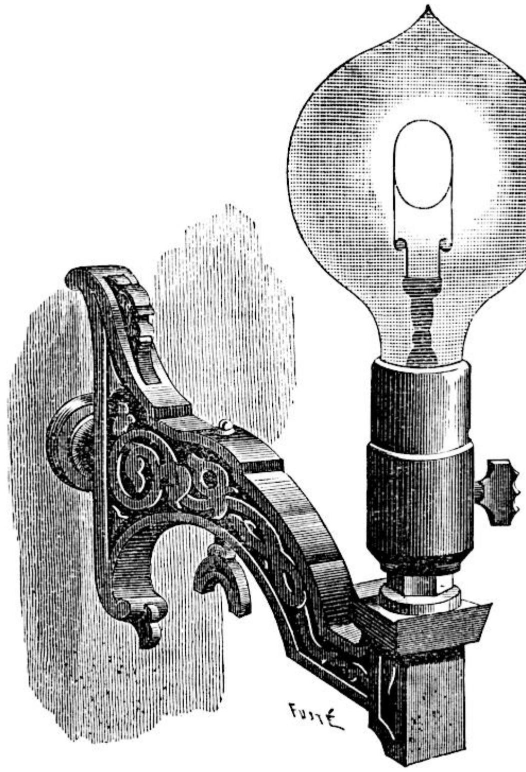
Before DTDeum

- Used Perl with SAX to read DTD declarations
- Hard to integrate with XSLT work
- Silently expanded parameter entities
- Needed cross-language (JS, Java, Rust...) support

Ideas

- Want/need DTD access in XSLT.
- Do not want to write it in Java, Rust, and C.
- Couldn’t find someone who had done this.
- Ask someone else to write it for me? But I don’t know what I want yet exactly.
- What to do?

Idea! Write it in XSLT!



Idea!

- Since it's already sheer lunacy,

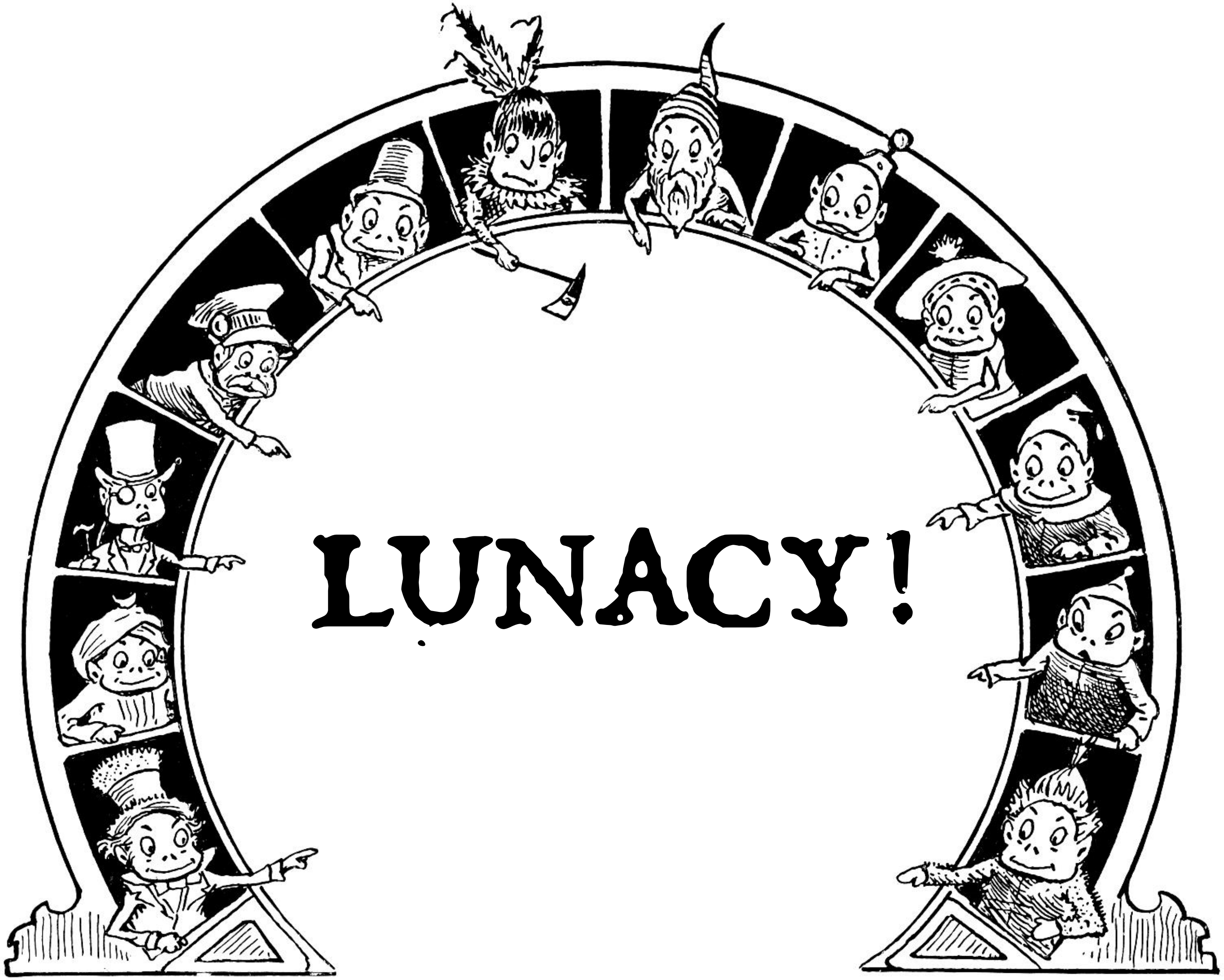
  inconceivable

  crazy

  stupid

  Let's use...

Regular Expressions!



LUNACY!

Regular Grammar

- The DTD grammar is defined with EBNF and so this is obviously easy, right?
- Constructs all look like <!ELEMENT *stuff* > so we can easily pick them out and processes them one by one.
- Oh, and processing instructions and comments. And conditional sections.

Declaration at a Time

- Parameter entities must end in the same context in which they start. This is illegal:

```
<!ENTITY % begin "<!ELEMENT" >
<!ENTITY % end ">" >
%begin; title (#PCDATA)* %end;
```

- We can always see the <! and the >

Matching Declarations

- Match one declaration: `<[^>]+>`
- Counter-example:

```
<!ATTLIST op
    gt CDATA #FIXED ">"
>
```

Token at a time

- So we could include parameter entities in the grammar and use invisible XML… what did you say?

```
<!ENTITY % q '"'>
<!ENTITY % type 'CDATA #FIXED "gossamer%q;'>
<!ATTLIST fairy wings %type;>
```

Gratuitous screaming fairies

Approach Taken

- A DTD contains a sequence of items;
- Each item is a processing instruction, comment, whitespace, parameter entity reference, or a declaration;
- Process one item at a time and recurse to look at the rest of the input.

One item at a time

- Identify the first token in the input;
- Call a function to handle the declaration;
- The function returns zero or more *decl* elements and also the remaining input still to parse;
- Recursively call the parser on the remaining input.

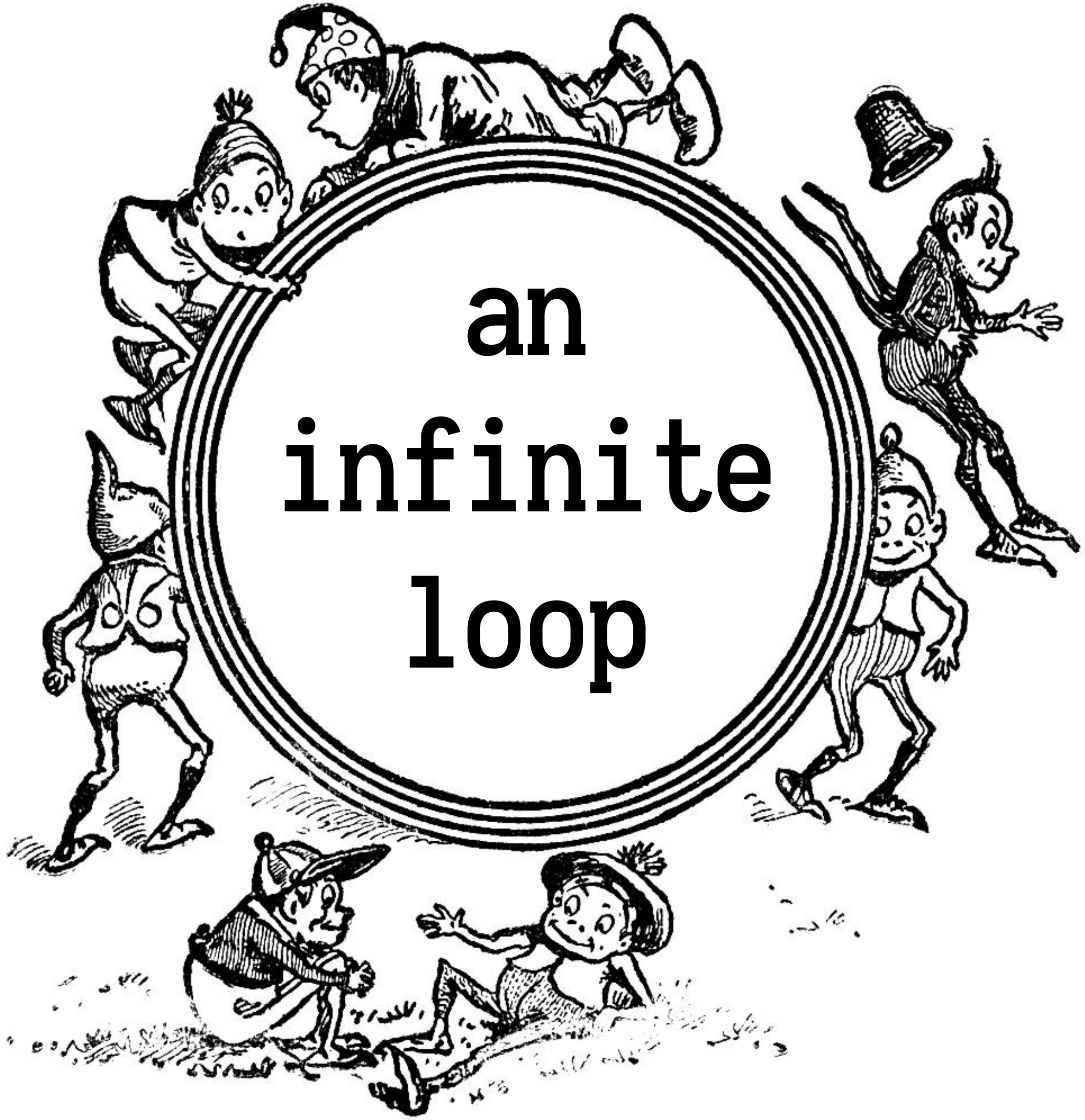
Let's Build it!

Sample regular expression

```
<xsl:variable name="regex" as="xs:string">
      ^ \s* <!ENTITY \s+
         % \s+ ({$XMLNAME}) \s+ (['"])(.*?)\2 \s*
     >\s*
</xsl:variable>
```

Declaration Parsing Function

- Receives as parameters: input, base URI, & results-so-far
- Uses a regular expression to find the end of the construct
- Calls a function to replace parameter entity references, using the declarations in *results-so-far*
- Constructs & returns new *decl* element(s) and the remaining unparsed input
- If a parsing function doesn't eat anything…



an infinite loop

Top level entry: dtd:parse-string()

- Called with string and base uri as parameters
- Ignores leading whitespace and comments
- Calls an appropriate handler function for the first token it finds and understands
- Then calls itself recursively for the rest, with the rest of the input as the string, and also with the results so far for parameter entity substitution

Another Day, Another Dragon

Too Much Nesting

- Each call to parse-string() has a new copy of the input (with one declaration removed)
- Works fine for small test cases, but fails for JATS, BITS, DocBook, etc.: uses too much memory.

Solution

- Solution: keep $input unchanged and pass an integer character position, a *cursor*.
- Now we only have a few bytes of memory (at a guess, 128 or so in Java, maybe 16 in Rust).
- Now DTDeum parses JATS and DocBook DTDs.

Other Grammas

Other Grammars

- Content models (up next)
- Attribute declarations
- Processing instructions with pseudo-attributes and NOTATION-declared targets (nope)
- Marked/Conditional sections <![IGNORE[supported with a recursive helper function
- Can ignore <![INCLUDE[and]]> (non-validating)

Content models

- Recursive function replaces one leaf particle in (parens) with «37=*» or «37=+» (where 37 is for the 37th particle, a * or + or ? is for occurrence), until there are no parens left.
- Then recursive function writes out the resulting string as elements.

Example

- (fairy, (goblin|(ogre,scream))*, (angel)+)
- (fairy, (goblin|«1=»)*,(angel)+)
- (fairy, «2=*»,(angel)+)
- (fairy, «2=*»,«3=+»)
- «4=»

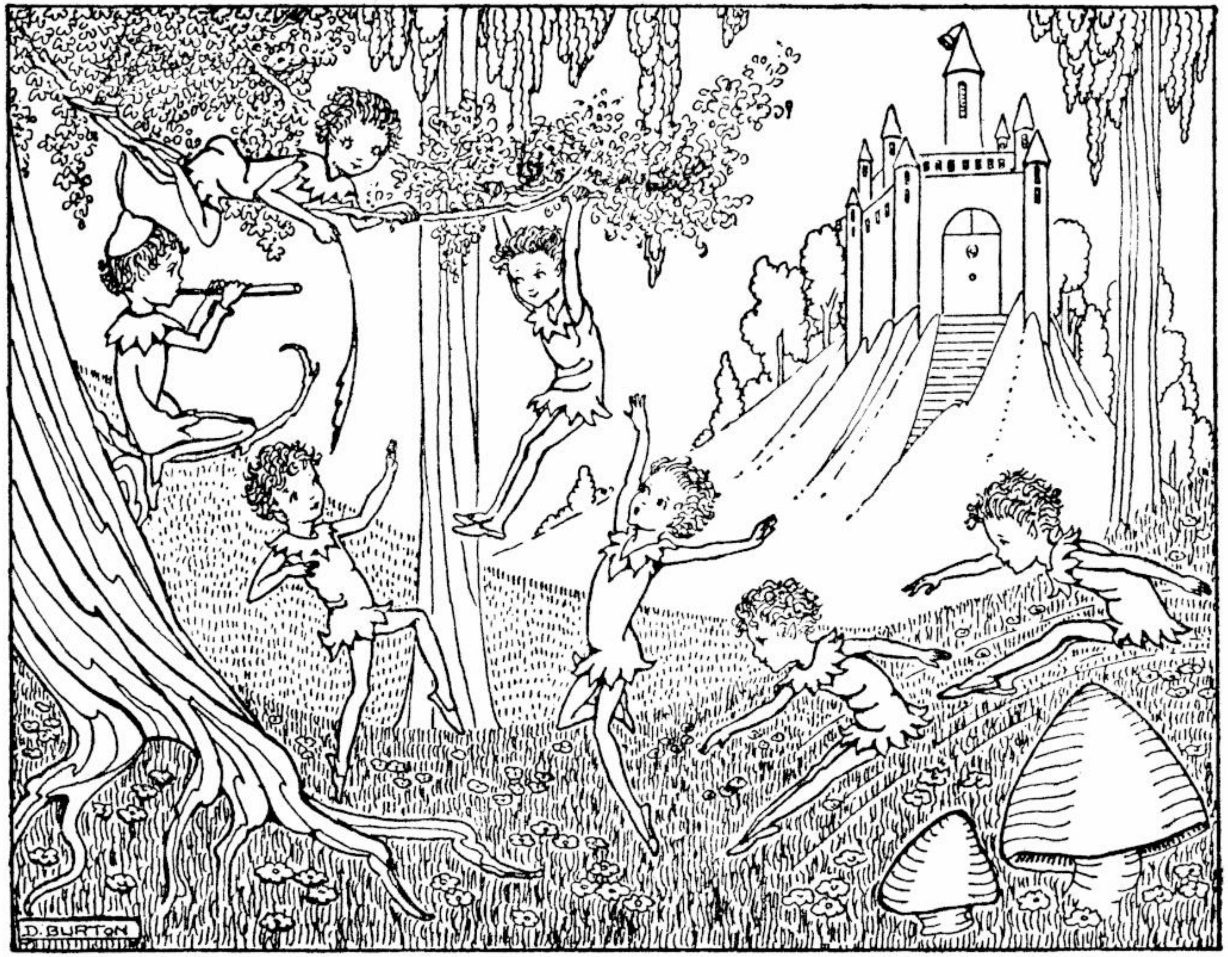
Processing the Content Model

```
<xsl:template mode="content-model" as="element(*)*"
  match=".[. instance of xs:string][contains(., '«')]">
    handle string before (apply-templates)
    handle «n=occ»
    handle string after (apply-templates)
</xsl:template>
```

Putting it all together

- Result is a sequence of *decl* elements
- Processes JATS, DocBook, etc
- Attribute lists not handled completely
- No diagrams yet
- TEI not handled ’cause it’s busted.

Forward to the Future



Future Work

- No diagrams yet, *attlist* not handled completely
- Possible function interface e.g. for XQuery or for hiding the *decl* elements, Mary Holstege style!
- Package it and put it on gitlab
- Entity resolver
- Improve errors (line numbers, missing files...)

Some lessons learned

- Elements or maps? If you use maps, be careful not to put XDM nodes into them - use copy-of(), to avoid keeping the containing document in memory;
- Use integer cursors rather than passing strings around;
- XPath regular expressions have some gotchas that are a pain; qt4 may improve this somewhat;
- DTD access inside XSLT is potentially very useful;
- XSLT is really cool. But we knew that already.

Happy Outcome

Acknowledgements

- A big thank-you to the reviewer who spotted an egregious mistake in the draft of the paper!
- Thank you also to Balisage for accepting this as a late-breaking paper;
- Thank you for listening.

Questions

https://gitlab.com/barefootliam/dtdeum

Liam Quin, Delightful Computing

https://www.delightfulcomputing.com/

Image Credits

- 1, 10, 21, El Mundo Ilustrado, 1883
- 4, Mother Goose, Ill. Arthur Rackham
- 7, Paracelsus, the Hermetic and Alchemical Writings of, Ed. A. E. Waite, 1894
- 12, 18, 24, Cox, Another Brownie Book
- 26, Andrew Lang, Blue Fairy Book
- 29 Callot's Etchings
- 35, 40 Little Ones' Budget
- 38, Photograph by the author at L'Europa Hotel, Montréal