What's new in 3.0 (XSLT/XPath/XQuery) (plus XML Schema 1.1)

Michael Kay
Saxonica

XSD 1.1

- Now a Proposed Recommendation
- Which means it's waiting for final approval by the W3C Advisory Committee and the Director
- Should be a clear run to the finish:
  - test suite exists
  - two implementations (Xerces, Saxon) pass all the tests

What's in XSD 1.1?

- Assertions
- Conditional Type Assignment
- Elements in multiple substitution groups
- Open content models
- Generalization of xs:all
- xs:override
- Various restrictions removed

What's not in XSD 1.1

- PrecisionDecimal data type
  - hotly fought issue
  - W3C rule is that changes require consensus, so if there is strong objection then the status quo holds

Most significant feature?

- In my view, assertions:
  - Change the game
  - Shamelessly borrowed from Schematron
  - Many constraints are better expressed using predicates than using a grammar

Impact?

- Two existing implementations
  - gives confidence
  - but they don't cover the whole space
- Another suspected implementation in the wings
- Take-up depends primarily on the verticals: FpML, XBRL, GIS, etc etc.
  - expect it to be slow

XPath 3.0
Functions and Operators

(For both XQuery 3.0 and XSLT 3.0)

Higher-order Functions

Functions are now first-class values
(a new kind of item)

Finally, XSLT and XQuery are fully functional programming languages!

Inline functions

```
let $sq :=
    function($i as xs:integer) as xs:integer {
        $i * $i
    }
```

Inline functions are expressions and can appear anywhere an expression is allowed.

Other expressions that return function items

- Function literals:
  - fn:abs#1, fn:max#2, my:func#3
- Partial application (currying):
  - string-join(?, ', ')
  - contains(?, ?, 'http://collation/case-blind')
- Run-time discovery:
  - function-lookup($name, $arity)

Functions that take functions as an argument

- fn:filter($function, $sequence)
- fn:map($function, $sequence)
- fn:map-pairs($function, $seq1, $seq2)
- fn:fold-left($function, $initial, $sequence)
- fn:fold-right($function, $initial, $sequence)

Properties of functions

- function-name($function)
- function-arity($function)

Use cases for higher-order functions

- Dynamic despatch mechanism
  - alternative to XSLT template rules
  - substitute for polymorphism
- Overcome limitations of XDM type system
- Reusable algorithms such as detection of cycles in a graph
- Reduce the need to write simple things using recursion

Other new functions

- trig/math functions: sin(), cos(), sqrt() etc
- analyze-string()
- format-date(), format-number(), generate-id(), unparsed-text() etc
  - moved from XSLT to common library
- head(), tail(), path()
- environment-variable(), uri-collection()
- parse(), serialize()

XSLT 3.0

- Streaming
- Packaging
- Other goodies

XSLT 3.0 “Goodies”

- xsl:try/catch (dynamic errors)
- xsl:evaluate (XPath expressions)
- xsl:iterate (a fold that looks like a for-each)
- extended pattern syntax
  - apply templates to atomic values
- xsl:merge (pre-sorted input files)
- declare type of initial context item

XSLT 3.0 Packaging

- Intended to allow separate compilation of modules
- Gives software engineering benefits for developing large stylesheets

Packages

- “Package” is a self-contained collection of modules that must declare its dependencies on other modules
- Controlled visibility of declarations
  - public, private, abstract, final
- Controlled override rules:
  - an overriding function or template must have a matching signature

Streaming

- General approach:
  - implementations always allowed to do streaming
  - define a subset of the language that is guaranteed streamable (if the processor supports this option)
  - streamability is a property of a mode (set of template rules); some documents may be processed using a streaming mode, others in a non-streaming mode

Current streamable subset

- Every template rule is allowed one downward selection
- Path analysis (data flow analysis) ensures that this takes into account variables and function calls
- Processor is required to compute all navigation paths in the streamed document and test this against a set of rules

Re-examining Streaming

- Current rules suffer from too little implementation experience or feedback
- Current rules assume too much about implementation strategy
- Current rules make it hard for users to understand why their code is (not) streamable
- Rules are very detailed and hard to debug
- Reviewing the strategy this coming week

The XSLT Maps proposal

- Motivations:
  - when streaming, you need more complex data structures to remember what you've seen in the document
  - extensibility of functions such as parse() and serialize()
  - support for JSON

XSLT Maps proposal

- A map is a new type of item
- Maps are immutable and have no identity
  - operations such as put() create a new map
- New syntax
  - constructor: map { “a” := “b” }
  - item type: map(keytype, valuetype)
- New functions:
  - get(), put(), contains(), keys(), size()

A map is a function

- Why?
  - allows $map(“key”) to get an entry
  - allows maps to be used wherever functions can be used, e.g. filter() and map() functions
  - economy of concepts

XSLT proposal for JSON

- Two new functions, parse-JSON() and serialize-JSON()
- Convert JSON to maps, not to XML
- Recognize JSON only at the boundaries (these two functions)
- Weak support for arrays (represented as maps from integers to values)