Functional XML: A preliminary sketch

Henry S. **Thompson**

25 November 2005

Abstract

Existing XML processing models are pipelines, controlled by pipeline descriptions which resemble shell scripts. Functional XML allows XML documents to specify their own processing explicitly, without losing the generality of the pipeline script approach.

Table of Contents

1. XML processing

XML processing is a heavily overloaded term, appealing as it does to a wide range of possible understanding of the 'meaning' of an XML document. At the base level, the XML Recommendation [XML] assigns a meaning to character streams associated with one of the XML family of media types in terms of a tree-structured document abstraction, whose detailed specification is given by the Infoset Recommendation [Infoset]. Applications of XML, i.e. particular XML vocabularies with an associated semantics, may in turn specify a further layer of meaning in terms of a mapping to/from some abstract data model. Examples of this include W3C XML Schema (schema components) [XSD], SVG (graphical objects) [SVG] and RDF (triples) [RDF].

Many W3C XML-related specifications can be understood as having a functional semantics, that is, as specifying a mapping from XML (infoset) to XML (infoset). XML Schema [XSD], XSLT [XSLT], XQuery [XQuery], XInclude [XIncl], XML Encryption [XEnc] and XML Signature [DSIG] are all in this category. Individual [WSDL] operations can also be understood in this way. There has recently been a significant growth in the range of tools available for controlling sequences of infoset-to-infoset mappings. These tools are usually described as specifying XML pipelines, and include at least the following:

- Sun's original W3C Pipeline Note [XPDL]
- Markup Technology's MT Pipeline [MTPL]
- Sean McGrath's XPipe [XPipe]
- Norm Walsh's SXPipe [SXPipe]
- Orbeon's XPL [XPL]
- 1060 Research's NetKernel [NetKernel]

These existing pipeline languages have a common core, in which XML processing is defined by a pipeline, which is itself an XML document. A pipeline specifies a sequence of high-level operations, drawn from an inventory such as the list above, to be chained together, one after another, each operating on the output of the one before. Some pipeline systems also provide operations at a lower level, allowing manipulation of parts of documents. Another common feature is provision for conditional processing. Here's an example of a simple pipeline specifying a sequence of inclusion, validation and styling:

```
<?xml version="1.0" encoding="utf-8"?>
<p0:pipeline xmlns:p0="http://www.w3.org/2002/02/xml-pipeline">
 <p0:processdef name="transform" definition="MT_XSLT_1.0"/>
 <p0:processdef name="include" definition="MT_XInclude"/>
 <p0:processdef name="validate" definition="MT_W3C_XML_Schema_1.0"/>
 <p0:process type="include">
  <p0:input label="$IN"/>
  <p0:output label="#i2.1"/>
 </p0:process>
 <p0:process type="validate">
  <p0:input label="#i2.1"/>
  <p0:input name="schema" label="po.xsd"/>
  <p0:output label="#i4.2"/>
 </p0:process>
 <p0:process type="transform">
  <p0:input label="#i4.2"/>
  <p0:input name="stylesheet" label="po.xsl"/>
  <p0:output label="$OUT"/>
```

```
  </p0:process>
</p0:pipeline>
```

2. An alternative, functional, perspective on XML processing

An alternative approach to XML processing is already in place in a somewhat fragmented and inconsistent way. Consider the following markup items which may be present in an XML document:

- the `xsi:schemaLocation` attribute on a document element
- the `xml-stylesheet` processing instruction/the `xsl:version` attribute on a (non-XSLT) document element
- the `http://www.w3.org/2001/04/xmlenc#` namespace

Each of these has a W3C Recommendation-based processing semantics -- a document with one of these markup items can be understood as saying, respectively:

- To validate me, use this schema document.
- To transform me, use this stylesheet.
- To decrypt me, you need this kind of key.

More recently, GRDDL [GRDDL] provides a way for a document to indicate, using a `data-view:interpreter` attribute, a transformation which will produce RDF statements. The presence of this attribute thus can be understood as saying "To understand me, use this stylesheet."

All of these markup items, in other words, provide information about ways that documents containing them might be processed. From a processing perspective, they can be understood as saying "Validate/transform/decrypt/understand me this way."

These mechanisms are neither systematic nor universal. The goal of Functional XML (hereafter f(X)) is to allow XML documents to provide processing information in a systematic and fully general way.

3. The f(X) approach

As noted above, the first-level semantics of an XML document serialisation is its own XML infoset. f(X) allows for the creation of XML documents which signal a second-level semantics *for themselves* in terms of one or more infoset-to-infoset mappings. It does this by specifying a compositional infoset-mapping interpretation for elements in the f(X) namespace, covering all the specifications mentioned above.

The names for the mappings covered by f(X) are chosen to describe the standards-based processing required. A mapping is understood to designate the *output* of the specified process. In the simplest cases, their *input* is the infoset designated in turn by their single child element. Taking schema validation and decryption as our starting point, we get the following examples:

Example 1. Simple f(X) example: decryption

```
<?xml version='1.0'?>
<fx:decrypt xmlns:fx="http://www.w3.org/2005/05/fx">
 <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
                MimeType='text/xml'>
  <CipherData>
   <CipherValue>A23B45C56 . . .</CipherValue>
  </CipherData>
 </EncryptedData>
</fx:decrypt>
```

Designates the infoset resulting from decrypting the ciphertext and parsing the resulting stream as XML.

Example 2. Simple f(X) example: validation

```
<?xml version='1.0'?>
<fx:validate xmlns:fx="http://www.w3.org/2005/05/fx">
 <purchaseOrder xmlns="http://www.example.com/PurchaseOrder"
                  xmlns:ad="http://www.example.com/Address"
                  orderDate="1999-10-20"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.example.com/PurchaseOrder po.xsd">

 <shipTo>
  <ad:name>Alice Smith</ad:name>
  <ad:street>123 Maple Street</ad:street>
  <ad:city>Mill Valley</ad:city>
  <ad:state>CA</ad:state>
  <ad:zip>90952</ad:zip>
 </shipTo>
 <billTo>
  <ad:name>Bill Gates</ad:name>
  <ad:street>123 Rich Guy Street</ad:street>
  <ad:city>Redmond</ad:city>
  <ad:state>WA</ad:state>
  <ad:zip>99999</ad:zip>
 </billTo>
 <comment>Hurry, my lawn is going wild!</comment>
 <items>
  <item partNum="872-AA">
   <productName>Lawnmower</productName>
   <quantity>1</quantity>
   <price>148.95</price>
   <comment>Confirm this is electric</comment>
  </item>
  <item partNum="926-AA">
   <productName>Baby Monitor</productName>
   <quantity>1</quantity>
   <price>39.98</price>
   <shipDate>1999-05-21</shipDate>
  </item>
 </items>
</purchaseOrder>
</fx:validate>
```

Designates the post-schema-validation infoset resulting from schema validity assessment of the basic infoset corresponding to the `purchaseOrder` element.

The simplicity and power of this approach, and the way in which it most clearly moves beyond the existing *ad hoc* signalling mechanisms mentioned above, become apparent once we actually compose multiple f(X) elements in a single document:

Example 3. Simple composition with f(X)

```
<?xml version='1.0'?>
<fx:validate xmlns:fx="http://www.w3.org/2005/05/fx">
 <fx:decrypt>
  <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
     MimeType='text/xml'>
   <CipherData>
    <CipherValue>A23B45C56 . . .</CipherValue>
   </CipherData>
  </EncryptedData>
 </fx:decrypt>
</fx:validate>
```

PSVI of decrypted document

But with respect to validation and decryption, the other order makes sense too:

Example 4. Another simple composition with f(X)

```
<?xml version='1.0'?>
<fx:decrypt xmlns:fx="http://www.w3.org/2005/05/fx">
 <fx:validate>
  <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
     MimeType='text/xml'>
   <CipherData>
    <CipherValue>A23B45C56 . . .</CipherValue>
   </CipherData>
  </EncryptedData>
 </fx:validate>
</fx:decrypt>
```

Decryption of schema-validated document

Indeed, validation before *and* after decryption is probably often what is wanted. That is, first we check that the encrypted data is valid per the XML Encryption namespace schema, then we decrypt, then we validate the result to check that *it's* OK.

Example 5. Richer composition

```
<?xml version='1.0'?>
<fx:validate xmlns:fx="http://www.w3.org/2005/05/fx">
 <fx:decrypt>
  <fx:validate>
   <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
      MimeType='text/xml'>
    <CipherData>
     <CipherValue>A23B45C56 . . .</CipherValue>
    </CipherData>
   </EncryptedData>
  </fx:validate>
 </fx:decrypt>
</fx:validate>
```

PSVI of decryption of schema-validated document

4. Generalizing f(X)

The outline of a simple functional language is emerging, but one of limited generality as far as the examples given above. Actually wrapping existing XML documents with f(X) elements to indicate preferred processing won't always work. The starting point may not be local, or may be read-only, or several alternative designations may be appropriate for alternative purposes. But the means to cover these cases is already there in principle, because we've already said we need an f(X) element for XInclude. Consider the following:

Example 6. f(X) with XInclude

```
<?xml version='1.0'?>
<fx:decrypt xmlns:fx="http://www.w3.org/2005/05/fx">
 <fx:include>
  <xi:include xmlns:xi="http://www.w3.org/2001/XInclude"
              href="encrypted.xml"/>
 </fx:include>
</fx:decrypt>
```

Functional equivalent of Example 1, “Simple f(X) example: decryption”

The designation of the `fx:include` element is the result of doing XInclude processing on the designation of its child element. Since the simple pattern above is likely to be very common, it can be abbreviated as follows:

Example 7. f(X) with XInclude, simplified

```
<?xml version='1.0'?>
<fx:decrypt xmlns:fx="http://www.w3.org/2005/05/fx">
 <fx:include href="encrypted.xml"/>
</fx:decrypt>
```

Simplified version of Example 6, “f(X) with XInclude”

The use of `fx:include` *allows* us to separate the statement of intended or desired designation from the core document, but does not require it.

5. Summary of f(X) so far

f(X) provides a means for specifying the desired designation of XML documents in a systematic and compositional way. It does so by specifying the designation of three basic classes of XML elements

`fx:include` elements	Designate the result of first interpreting the `href`, `xpointer` and other XInclude attributes per the XInclude spec., then applying these f(X) rules to the resulting infoset;
other elements in the f(X) namespace	Designate the result of the mapping specified by their name applied to the designations of their children;
all other elements	Designate themselves, that is, their ordinary infosets, except in-so-far as they contain elements in the f(X) namespace, which are interpreted per the above two clauses.

6. Completing basic f(X)

A few things need to be added to cover the intended basic functionality.

It should be possible to *prevent* the special treatment f(X) specifies for the first two classes of elements above -- f(X) provides the `fx:sic` element for this purpose:

Example 8. 'Quoting' with fx:sic

```
<?xml version='1.0'?>
<fx:sic xmlns:fx="http://www.w3.org/2005/05/fx">
 <fx:include href="encrypted.xml"/>
</fx:sic>
```

Designates a single-element `fx:include` document.

Also, we provide a `sic` attribute on `fx:include`, which defaults to ***false***, but which if ***true*** blocks recursive f(X) processing of the inclusion target.

Finally, we need a way of specifying more than one input infoset and, for those specifications which require (or allow) it, parameters. f(X) allows for parameters via attributes on the relevant f(X) elements, and allows additional children where appropriate to designate additional input infosets. For example, for `fx:transform` (XSLT) we allow a second child to directly provide the stylesheet:

Example 9. Primary and secondary input infosets

```
<?xml version='1.0'?>
<fx:transform xmlns:fx="http://www.w3.org/2005/05/fx">
  <fx:include href="po.xml"/>
  <fx:include href="po.xsl"/>
</fx:transform>
```

Since infosets such as stylesheets and schema documents are so often static, it also makes sense to allow them to appear as attributes on the relevant f(X) element:

Example 10. Static infosets as attributes

```
<fx:transform xmlns:fx="http://www.w3.org/2005/05/fx">
            stylesheet="po.xsl">
 <fx:validate schemaDocuments="po.xsd address.xsd">
  <fx:include href="po.xml"/>
 </fx:validate>
</fx:transform>
```

Style the result of validation, using static resources for schema documents and stylesheet

Finally we need to list at least a preliminary set of built-in f(X) designators for each public specification which can be understood as defining XML-to-XML functions:

fx:dtdValidate	Validated W3C XML [XML]
fx:validate	W3C XML Schema [XSD]
fx:transform	W3C XSLT (v.1 [XSLT] or v.2 [XSLT2], depending on stylesheet)
fx:query	W3C XML Query [XQuery]
fx:encrypt	W3C XML Encryption [XEnc]
fx:decrypt	W3C XML Encryption [XEnc]
fx:sign	W3C XML Signature [DSIG]
fx:verify	W3C XML Signature [DSIG]
fx:include	W3C XML Include [XIncl]
fx:interpret	GRDDL [GRDDL]

7. Beyond basic f(X): Choosing and binding

As mentioned above, some existing pipeline languages allow for conditional processing. If it is judged appropriate to include something like this in f(X), it can be done easily, following the model of XSLT's `choose`:

```
<fx:case>
 <fx:when test="boolean-valued XPath expression">
  infoset to test
  result infoset if test succeeds </fx:when>
 . . .
 <fx:otherwise>
  result infoset if no test succeeds
 </fx:otherwise>
</fx:case>
```

Example 11. Using fx:case for conditional processing

```
<fx:case>
 <fx:when test="/root/@version > 3">
  <fx:include href="doc.xml"/>
  <fx:validate schemaDocuments="current.xsd">
   <fx:include href="doc.xml"/>
  </fx:validate>
 </fx:when>
 <fx:otherwise>
  <fx:validate schemaDocuments="stale.xsd">
   <fx:include href="doc.xml"/>
  </fx:validate>
 </fx:otherwise>
</fx:case>
```

Choosing a schema document based on an XPath expression test

`fx:when` has a `test` attribute for an XPath expression and two infoset arguments. The first is the infoset to test with the XPath expression, the second the result if the test is satisfied.

Clearly if interpreted literally we have a lot of potential for wasted effort here with respect to the `doc.xml` resource. There are two possibile ways f(X) could address this. It could do nothing beyond noting that implementors may detect and optimize such cases, or it could provide for explicit binding of infosets to variables, which can then be referenced by XPath expressions or an `fx:infoset` element.

Once again we'll take XSLT as our model, using `xs:variable` to bind names to constructed or denoted-by-URI infosets:

```
<fx:with>
 <fx:variable name="var name">
  f(X) designating an infoset
 </fx:variable>
 <fx:variable name="another name
              href="URI for infoset"/>
 . . .
</fx:with>
```

Example 12. Binding infosets to variables

```
<fx:with>
 <fx:variable name="doc" href="doc.xml"/>
 <fx:case>
  <fx:when test="$doc/root/@version > 3">
   <fx:validate schemaDocuments="current.xsd">
    <fx:infoset expr="$doc"/>
   </fx:validate>
  </fx:when>
  <fx:otherwise>
   <fx:validate schemaDocuments="stale.xsd">
    <fx:infoset expr="$doc"/>
   </fx:validate>
  </fx:otherwise>
 </fx:case>
</fx:with>
```

Explicit binding to avoid extra work

The provision of an explicit binding mechanism would clearly be of use, particularly since in cases where testing needs to be done on the result of some more or less complex composition of f(X) elements it would enable the concise specification of dependencies which would otherwise require egregious duplication of structure. However there's a real question as to whether this opens up too many uncertainties. In particular the introduction of variable binding into pure functional programming languages is known to have a significant impact on overall computational complexity. . .

8. Drawing the obvious parallel

At this point f(X) is beginning to look a lot like a transcription of a traditional functional programming language such as Scheme into XML. The previous example Example 12, “Binding infosets to variables”, for instance, is just a transcription of something like this expression, which is meant to be Scheme plus a bit of syntactic sugar for XPath expressions and URIs:

Example 13. Compact notation

```
(let ((doc {doc.xml}))
 (if (gt |$doc/root/@version| 3)
    (fx:validate doc {current.xsd})
   (fx:validate doc {stale.xsd})))
```

f(X) is really just XMLised Scheme. . .

Given the history of negative reactions from the Web community to Scheme, it's not clear that promoting this compact syntax would be a good idea, but it is clearly both a fruitful source of insight and a good starting point for any formal characterisation of the semantics of f(X).

9. Implementation strategy

Implementations are already available for a number of pipeline languages. It is straightforward to translate basic f(X) into many of these, particularly as long as all the f(X) elements in the document have at most a single child. Multiple

children, as will as `fx:case` and `fx:with`, will require a bit more work, and may not be straightforwardly possible in all existing languages.

10. Conclusion

f(X) provides a novel approach to XML processing, an alternative way of conceptualizing in terms of function composition what is currently largely understood in terms of pipelines. It amounts to a functional programming language whose primary data objects are complete XML documents, and as such is qualititatively distinct from such related efforts as XQuery [XQuery], XMLambda [XMLambda], Echo [Echo] or Links [Links].

Acknowledgements

The work reported here was initiated by a discussion with Tim Berners-Lee, who also first used the phrase "functional XML" in my hearing. The basic direction was first suggested by Richard Tobin.

Bibliography

[DSIG] *XML-Signature Syntax and Processing* *[http://www.w3.org/TR/xmldsig-core/]*, Donald Eastlake, Joseph Reagle and David Solo, eds.

[Echo] *Echo XML Data Processing Language* *[http://docs.xmlecho.org/echo-spec.html]*, Bill Lindsey.

[GRDDL] *Gleaning Resource Descriptions from Dialects of Languages (GRDDL)* *[http://www.w3.org/TeamSubmission/grddl/]*, Dominique Hazaël-Massieux and Dan Connolly

[Infoset] *XML Information Set (Second Edition)* *[http://www.w3.org/TR/xml-infoset]*, John Cowan and Richard Tobin eds.

[Links] *Links: Linking Theory to Practice for the Web* *[http://homepages.inf.ed.ac.uk/wadler/links/]*, Philip Wadler.

[MTPL] *Re-Interpreting the XML Pipeline Note* *[http://www.markup.co.uk/XML2003.html]*, Henry S. Thompson.

[NetKernel] *Building Robust Heterogeneous Asynchronous XML Pipelines* *[http://www.idealliance.org/proceedings/xml04/papers/293/293.html]*, Peter Rodgers.

[RDF] *Resource Description Framework (RDF): Concepts and Abstract Syntax* *[http://www.w3.org/TR/rdf-concepts/]*, Graham Klyne and Jeremy J. Carroll, eds.

[SVG] *SVG* *[http://www.w3.org/TR/SVG/]*, Jon Ferraiolo, Jun Fujisawa and Dean Jackson, eds.

[SXPipe] *SXPipe: Simple XML Pipelines* *[http://norman.walsh.name/2004/06/20/sxpipe]*, Norm Walsh.

[WSDL] *Web Services Description Language 2.0* *[http://www.w3.org/TR/wsdl20/]*, Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman and Sanjiva Weerawarana, eds.

[XEnc] *XML Encryption Syntax and Processing* *[http://www.w3.org/TR/xmlenc-core/]*, Donald Eastlake and Joseph Reagle, eds.

[XIncl] *XML Inclusions (XInclude) Version 1.0* *[http://www.w3.org/TR/xinclude/]*, Jonathan Marsh and David Orchard eds.

[XML] *Extensible Markup Language (XML) version 1.1 [http://www.w3.org/TR/XML11/]*, Tim Bray, C. M. Sperberg-McQueen et al., eds.

[XMLambda] *XMLambda: A functional language for constructing and manipulating XML documents [http://www.cartesianclosed.com/pub/xmlambda/]*, Erik Meijer and Mark Shields.

[XPDL] *XML Pipeline Definition Language Version 1.0 [http://www.w3.org/TR/xml-pipeline/]*, Norman Walsh and Eve Maler.

[XPL] *XML Pipeline Language (XPL) Version 1.0 [http://www.w3.org/Submission/xpl/]*, Erik Bruchez and Alessandro Vernet.

[XPipe] *XPipe - An XML Processing Methodology [http://www.idealliance.org/papers/xml2001/papers/html/05-01-02.html]*, Sean McGrath.

[XQuery] *XQuery 1.0: An XML Query Language [http://www.w3.org/TR/xquery/]*, Don Chamberlin et al., eds.

[XSD] *XML Schema Part 1: Structures [http://www.w3.org/TR/xmlschema-1/]*, Henry S. Thompson et al., eds.

[XSLT] *XSL Transformations (XSLT) Version 1.0 [http://www.w3.org/TR/xslt]*, James Clark ed.

[XSLT2] *XSL Transformations (XSLT) Version 2.0 [http://www.w3.org/TR/xslt20/]*, Michael Kay ed.

Biography

Henry S. **Thompson**
University of Edinburgh
ICCS/HCRC, Division of Informatics [http://www.iccs.inf.ed.ac.uk/]
Edinburgh
United Kingdom
Markup Technology [http://www.markup.co.uk/]
World Wide Web Consortium [http://www.w3.org/]

Henry S. Thompson divides his time between the School of Informatics at the University of Edinburgh, where he is Reader in Artificial Intelligence and Cognitive Science, based in the Language Technology Group of the Human Communication Research Centre, and the World Wide Web Consortium (W3C), where he works in the XML Activity.

He received his Ph.D. in Linguistics from the University of California at Berkeley in 1980. His university education was divided between Linguistics and Computer Science, in which he holds an M.Sc. While still at Berkeley he was affiliated with the Natural Language Research Group at the Xerox Palo Alto Research Center, where he participated in the GUS and KRL projects. His research interests have ranged widely, including natural language parsing, speech recognition, machine translation evaluation, modelling human lexical access mechanisms, the fine structure of human-human dialogue, language resource creation and architectures for linguistic annotation. His current research is focussed on articulating and extending the architectures of XML.

He was a member of the SGML Working Group of the World Wide Web Consortium which designed XML, is the author of the XED, the first free XML instance editor and co-author of the LT XML toolkit and is currently a member of the XML Core and XML Schema Working Groups of the W3C, and has recently been elected to the W3C TAG (Technical Architecture Group). He is lead editor of the Structures part of the XML Schema W3C Recommendation, for which he co-wrote the first publicly available implementation, XSV. He has presented many papers and tutorials on SGML, DSSSL, XML, XSL and XML Schemas in both industrial and public settings over the last eight years.