XProc in Action - Revisited

Overview

XML Processing Pipelines are by no means new, they've been around in many forms and for as long as there have been tools to process XML. More often than not they have been created out of high-level tools that are normally used to string various disparate programming task together. A classic example being Apache Ant. However, tools like Ant don't create a pipeline as such, there is no true concept of connecting inputs and outputs.

Formerly, the most recognised example of 'proper' XML processing pipelines was Apache Cocoon. At the heart of Cocoon is the 'sitemap' which is a declarative language that allows the developer to describe sequences of XML processing operations. The pipelines sit within a larger framework that maps them to HTTP requests by way of URI pattern matching.

The W3C's XML Pipeline Language - XProc, is an attempt at defining a standard for connecting a wide range of both basic and complex XML processing steps into a single declarative XML language that adheres to the central concept of pipeline processing. In XProc every step has defined *input* and *output* ports which can themselves be connected to in-line or external data sources as well as the results of other processing steps. The flow within a pipeline can be largely sequential, following the order of steps in the pipeline (document order) but, more interestingly, and usefully, a step can have many input and/or output ports and these ports can accept inputs from many other steps within the pipeline, leading to very complex and advanced workflows.

Why XProc is Important

As has been already suggested, XProc has great flexibility in how processing steps are connected together and when that is married to the range of built-in steps plus the ability to create new compound steps and the provision for extensions, XProc can be seen as a very powerful tool for XML developers.

One of the re-occurring themes in projects where XML is being processed is the impedance miss-match between the content to be processed and the programming language being used to process it. Many yards of code are written in order to read documents, instantiate transformers, serialise results etc, etc. XProc neatly manages to abstract away all, or at least most, of the complexity that ensues when handling XML, and it does so in away that is meaningful to an XML developer rather than to a procedural/object-oriented programmer. This is the real power and potential of XProc, it places the XML developer in the driving seat.

Potential Use Cases

There are many cases where XProc can be utilised in XML processing tasks ranging from environment set-up, content transformation, business workflows unit and system testing too.

Environment Set-up

Where systems provide an API that allows programmatic access to administration functions, for example the MarkLogic Server Admin API, it is possible to write simple XQuery main modules that can be invoked from the pipeline. Norman Walsh's XML Calabash has MarkLogic specific extensions for invoking queries within a specific host. Forests, Databases, App Servers and the like can be created, configured, reconfigured and finally deleted. This has uses in not only replicating environments for production but also in the set-up and tear-down of individual test environments. Obviously the real action goes on within the XQuery code but it's the pipeline that can string-together various modular steps, and it is the modular approach that makes the use of XProc desirable.

~~The initial work in this area was biased towards executing a few XQueries with many function calls. More recently work has started on creating an XProc language binding for the Admin API where each function is a single step declaration.~~

Content Loading

Content can can be presented for loading in many different ways. It may reside on the file system or be accessible from a Web Service of some sort as single huge documents with many fragments to be ingested, or vast numbers of individual documents. An XProc pipeline can handle all these scenarios by use of `load`, `directory-list` and `http-request` steps. The sequence of documents to be loaded can then have any number of processing step applied that may carry-out mundane tasks like resolving relative URIs, adding unique identifiers, chunking large documents, all the way to complex transforms.

Finally, once again using Mark Logic specific extensions, Calabash can insert these documents, or fragments of documents, into the desired database via an XDBC connection. This is very similar to what can be achieved using RecordLoader with the exception that many additional processing steps can be performed, with XProc, giving greater flexibility outside of MarkLogic.

Content Processing

With a combination of the techniques used in the two previous use-cases, it is also possible to create content processing tasks that work in just the same way that Corb does by identifying a collection of document URIs and then invoking an XQuery against each document which those URIs identify. It also goes without saying that many of the same processes can be run both client and server-side and with any number of steps.

Content Workflows

Any of the scenarios described here can, in general, be regarded as content workflows in one form or another. However, the particular branch of content processing being described here involves not only transformations but business logic too.

~~In the example where aggregating content from different sources requires matching data from one source to data that already resides within a database it is often necessary to uses traditional search techniques to identify possible candidates for matching before applying matching algorithms. Here, logic is required to determine two different courses of action depending on the number of candidates found and the resulting matches.~~

Unit & System Testing

There are three types of XProc documents that can be created; Pipelines, Step Declarations and Libraries. The first two are pipeline descriptions, which can be evaluated by a pipeline processor, whereas the third contains a collection of step declarations and can only be imported into a pipeline or step declaration.

Breaking-down pipelines into discrete and re-usable modules that are defined within Library documents allows greater flexibility for both re-use and testing. It is relatively straight forward to build test pipelines that target individual steps, feeding known inputs and matching the results to expected outputs. This can form the basis of both Unit Tests and System Tests.

Implementations

There are a number of XProc processors in development at present, most are almost complete as they strive to meet the requirements for the test suite.

- Norman Walsh's XML Calabash
- EMC's Calumet
- James Fuller's XProcxq

These are the main implementations with some others that are also either in development or currently dormant. Where both Calabash and Calumet are Java applications, XProcXQ runs within the eXist database and is written in XQuery. My own, partial, implementation of an XProc processor was an experiment called 'Half-pipe' that used XSLT 2.0 to compile an XProc pipeline into an XSLT transform that could be applied to the source document(s). I believe that a similar technique is used within the MarkLogic's Information Studio Flows.

I've made extensive use of Calabash over the last three years, or so, and I've also used Calumet for one particular task that required PDF output from XSL-FO documents. ~~A number of Calabash's dependencies are proprietary applications that require software licenses.~~

~~In general, both Calabash and Calumet work well, however, Calabash does have gremlins in the corners but that is to be expected of an application that is still essentially in beta testing.~~

The Future

XProc is now an official W3C Recommendation.

One interesting aspect of the XProc design is that the execution of the pipeline steps is purely implementation dependent and makes no demands upon the order of execution. This opens the possibility for parallel execution and and potentially streaming of large volumes of content. After all, in XProc, an input source can be a sequence of documents and there's nothing to say that that source cannot be a connection to a Enterprise Messaging System.

Once you start talking about enterprise level systems it not long before people start talking about process orchestration which, at present, is a concept that is outside the scope of XProc. This being so, it hasn't stopped people considering how they might specify a group of steps that must run in parallel or may be that they execute at a specific time or time intervals. One area that I've been looking into is the uses of SMIL Timesheets, that provide out-of-line declarations for timing and synchronization. The Synchronized Multimedia Integration Language (SMIL) is normally associated with multimedia and SVG but the concept of declaring the 'active phase' (start, duration, end and repetition) of a pipeline step is not an unreasonable match to that of animating a graphic primitive within an illustration. It remains to be seen whether this is a truly viable solution as the concept hasn't progressed beyond the napkin-sketch phase.