XProc: An XML Pipeline Language

Norman Walsh
Sun Microsystems, Inc.

Background

XProc Development
Working Group Goals
Goals (Continued)
Use Cases
XML Components
Hasn't this been done already?
Design Goals

XProc Development

- W3C XML Processing Model Working Group started in late 2005.
- Many familiar names: Erik Bruchez, Andrew Fang, Paul Grosso, Rui Lopes, Murray Maloney, Alex Milowski, Michael Sperberg-McQueen, Jeni Tennison, Henry Thompson, Richard Tobin, Alessandro Vernet, Norman Walsh, Mohamed Zergaoui
- Chaired by yours truly.
- We expect to be finished with XProc before our charter expires on 31 Oct 2007.

Working Group Goals

According to its charter, the goals of the XML Processing Model Working Group are to develop two Recommendation Track documents:

1. **An XML Processing Language which answers the following questions:**
   a. **What is to be done to a given document or a set of documents by a given sequence of given XML processors?**
   b. **How are exceptions handled during processing?**

Goals (Continued)

1. An XML Processing Model which answers the following questions:
   a. Which if any of the transformations signalled by aspects of an XML document should be performed, and in what order?
   b. How can an author, consumer, or application guide this process?
   c. In the absence of any guidance, what default processing, if any, should be done in what circumstances?

Use Cases

From *XML Processing Model Requirements and Use Cases*:

- **Apply a Sequence of Operations**
- **XInclude Processing**
- **Parse/Validate/Transform**
- **Document Aggregation**
- **Single-file Command-line Document Processing**
- **Multiple-file Command-line Document Generation**
- **Extracting MathML**
- **Style an XML Document in a Browser**
- **Run a Custom Program**
- **XInclude and Sign**
- **…**

XML Components

Hasn't this been done already?

Well, yes: Apache Ant, Cocoon Sitemaps, GNU JAXP Library: Package gnu.xml.pipeline, Jelly : Executable XML, MT Pipeline Overview, NetKernel - Service Oriented MicroKernel and XML Application Server, Oracle XML Developer's Kit Home, Re-Interpreting the XML Pipeline Note: Adding Streaming and On-Demand Invocation, Schemachine (a pipelined Xml validation framework), ServingXML, smallx: Project Home Page, Strawman: bringing the framework within the schemas, SXPipe: Simple XML Pipelines, Xerces Native Interface, XML-ECHO, XML Pipeline Definition Language Version 1.0, XML Pipeline Language (XPL) Version 1.0 (Draft), XPipe

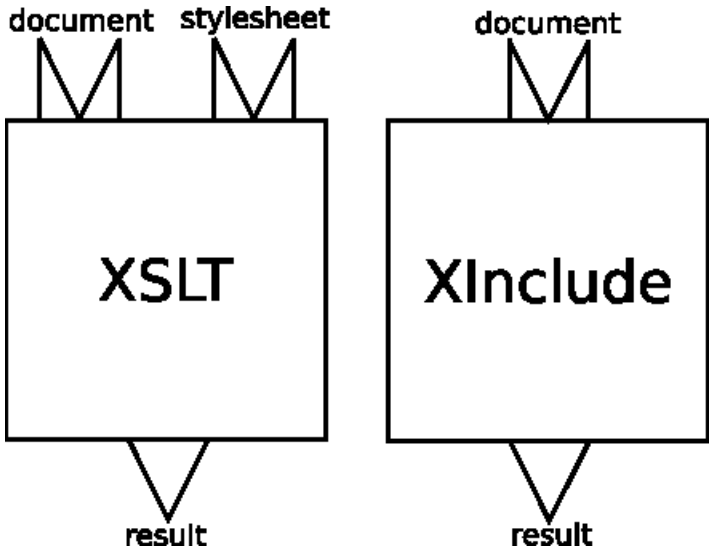
Design Goals

- Standardization, not design by committee
- Able to support a wide variety of components
- Prepared quickly
- Few optional features
- Relatively declarative
- Amenable to streaming
- “The simplest thing that will get the job done.”

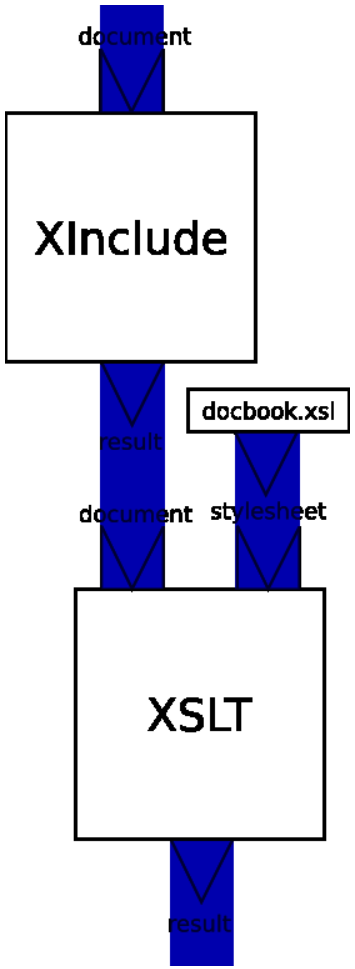
Pipeline Concepts

Pipeline Steps
The Pipeline Analogy
What flows through pipes?
Steps can be Grouped into Pipelines
Pipelines are Steps

Pipeline Steps



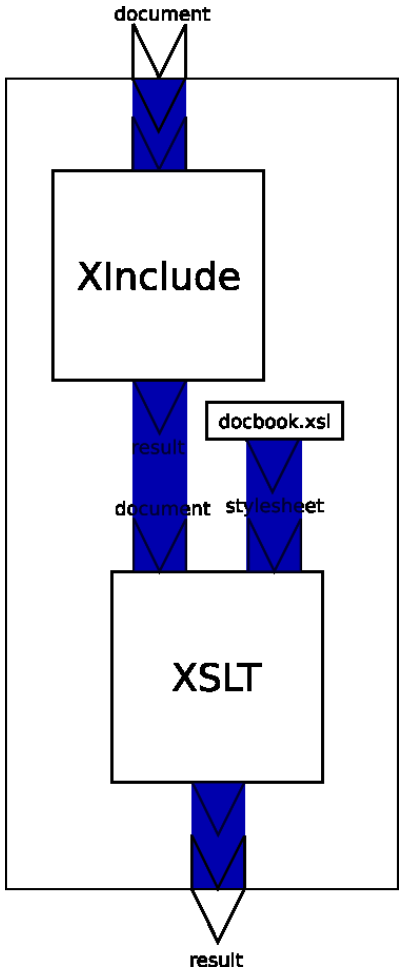
The Pipeline Analogy



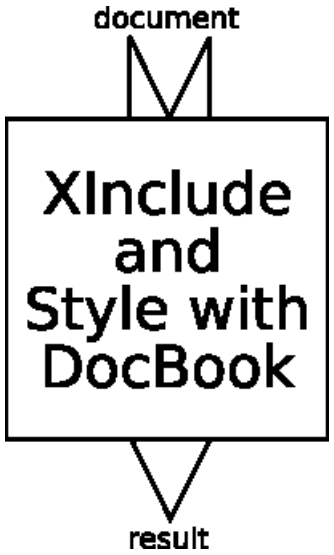
What flows through pipes?

- **The “water” of our pipelines are XML documents.**
- **In the specific case of XProc, we mean documents conceptually:**
  - **Not elements or nodes.**
  - **Not XDMs or Infosets or PSVIs (or rather, any of those.)**

Steps can be Grouped into Pipelines



Pipelines are Steps



XProc Pipelines

A Little Terminology
Declaring Steps
Using Steps
Anatomy of (an Atomic) Step
Anatomy of a Step (2)
Anatomy of a Step (3)
Anatomy of a Step (4)
Anatomy of an Input
…

A Little Terminology

- **Steps have named “ports”.**
- **The names of the ports on any given step are a fixed part of its signature.**
- **Inputs and outputs bind document streams to ports.**
- **Steps have a name and a type.**

Declaring Steps

```
<p:declare-step type="p:xinclude">
  <p:input port="source" sequence="no"/>
  <p:output port="result" sequence="no"/>
</p:declare-step>

<p:declare-step type="p:xslt2">
  <p:input port="source" sequence="yes"/>
  <p:input port="transform" sequence="no"/>
  <p:output port="result" sequence="no"/>
  <p:output port="secondary" sequence="yes"/>
  <p:option name="initial-mode"/>
  <p:option name="template-name"/>
  <p:option name="allow-version-mismatch" value="yes"/>
  <p:option name="output-base-uri"/>
  <p:option name="allow-collections" value="yes"/>
</p:declare-step>
```

Using Steps

```
<p:xinclude name="xinc">
  <p:input port="source">
    <p:document href="doc.xml"/>
  </p:input>
</p:xinclude>

<p:xslt2 name="xform">
  <p:input port="source">
    <p:pipe step="xinc" port="result"/>
  </p:input>
  <p:input port="stylesheet">
    <p:document href="docbook.xsl"/>
  </p:input>
</p:xslt2>
```

Anatomy of (an Atomic) Step

```
<p:xslt2 name="xform">
  <p:input port="source">
    <p:pipe step="otherstep" port="result"/>
  </p:input>
  <p:input port="stylesheet">
    <p:document href="docbook.xsl"/>
  </p:input>
  <p:option name="allow-collections" value="no"/>
  <p:option name="initial-mode" select="$imode"/>
  <p:parameter name="home" value="http://example.com/"/>
</p:xslt2>
```

Anatomy of a Step (2)

```
<p:xslt2 name="xform">
  <p:input port="source">
    <p:pipe step="otherstep" port="result"/>
  </p:input>
  <p:input port="stylesheet">
    <p:document href="docbook.xsl"/>
  </p:input>
  <p:option name="allow-collections" value="no"/>
  <p:option name="initial-mode" select="$imode"/>
  <p:parameter name="home" value="http://example.com/"/>
</p:xslt2>
```

Anatomy of a Step (3)

```
<p:xslt2 name="xform">
  <p:input port="source">
    <p:pipe step="otherstep" port="result"/>
  </p:input>
  <p:input port="stylesheet">
    <p:document href="docbook.xsl"/>
  </p:input>
  <p:option name="allow-collections" value="no"/>
  <p:option name="initial-mode" select="$imode"/>
  <p:parameter name="home" value="http://example.com/"/>
</p:xslt2>
```

Anatomy of a Step (4)

```
<p:xslt2 name="xform">
  <p:input port="source">
    <p:pipe step="otherstep" port="result"/>
  </p:input>
  <p:input port="stylesheet">
    <p:document href="docbook.xsl"/>
  </p:input>
  <p:option name="allow-collections" value="no"/>
  <p:option name="initial-mode" select="$imode"/>
  <p:parameter name="home" value="http://example.com/"/>
</p:xslt2>
```

Anatomy of an Input

A `p:input` identifies input to a *port*; its subelements identify a document or sequence of documents:

- **`<p:document href="`*`uri`*`"/>` reads input from a URI.**
- **`<p:inline/>` provides the input as literal content in the pipeline document.**
- **`<p:pipe step="`*`stepName`*`" port="`*`portName`*`"/>` reads from a readable port on some other step.**
- **`<p:empty/>` is an empty sequence of documents.**

Anatomy of a Step (5)

```
<p:xslt2 name="xform">
  <p:input port="source">
    <p:pipe step="otherstep" port="result"/>
  </p:input>
  <p:input port="stylesheet">
    <p:document href="docbook.xsl"/>
  </p:input>
  <p:option name="allow-collections" value="no"/>
  <p:option name="initial-mode" select="$imode"/>
  <p:parameter name="home" value="http://example.com/"/>
</p:xslt2>
```

Anatomy of a Step (6)

```
<p:xslt2 name="xform">
  <p:input port="source">
    <p:pipe step="otherstep" port="result"/>
  </p:input>
  <p:input port="stylesheet">
    <p:document href="docbook.xsl"/>
  </p:input>
  <p:option name="allow-collections" value="no"/>
  <p:option name="initial-mode" select="$imode"/>
  <p:parameter name="home" value="http://example.com/"/>
</p:xslt2>
```

Anatomy of a Compound Step

```
<p:group name="xincxform">
  <p:output port="result">
    <p:pipe step="xform" port="result"/>
  </p:output>
  <p:xinclude name="xinc">
    <p:input port="source">
      <p:pipe step="precstep" port="result"/>
    </p:input>
  </p:xinclude>
  <p:xslt2 name="xform">
    <p:input port="source">
      <p:pipe step="xinc" port="result"/>
    </p:input>
    <p:input port="stylesheet">
      <p:document href="docbook.xsl"/>
    </p:input>
  </p:xslt2>
</p:group>
```

Anatomy of a Compound Step (2)

```
<p:group name="xincxform">
  <p:output port="result">
    <p:pipe step="xform" port="result"/>
  </p:output>
  <p:xinclude name="xinc">
    <p:input port="source">
      <p:pipe step="precstep" port="result"/>
    </p:input>
  </p:xinclude>
  <p:xslt2 name="xform">
    <p:input port="source">
      <p:pipe step="xinc" port="result"/>
    </p:input>
    <p:input port="stylesheet">
      <p:document href="docbook.xsl"/>
    </p:input>
  </p:xslt2>
</p:group>
```

Simplify with Defaults

```
<p:group name="xincxform">
  <p:output port="result">
    <p:pipe step="xform" port="result"/>
  </p:output>
  <p:xinclude name="xinc">
    <p:input port="source">
      <p:pipe step="precstep" port="result"/>
    </p:input>
  </p:xinclude>
  <p:xslt2 name="xform">
    <p:input port="source">
      <p:pipe step="xinc" port="result"/>
    </p:input>
    <p:input port="stylesheet">
      <p:document href="docbook.xsl"/>
    </p:input>
  </p:xslt2>
</p:group>
```

Simple Group

```
<p:group>
  <p:output port="result"/>

  <p:xinclude/>

  <p:xslt2>
    <p:input port="stylesheet">
      <p:document href="docbook.xsl"/>
    </p:input>
  </p:xslt2>
</p:group>
```

Making a Pipeline

```
<p:pipeline name="main">
  <p:input port="document"/>
  <p:input port="stylesheet"/>
  <p:output port="result"/>

  <p:xinclude>
    <p:pipe step="main" port="document"/>

  <p:xslt2>
    <p:input port="stylesheet">
      <p:pipe step="main" port="stylesheet"/>
    </p:input>
  </p:xslt2>
</p:pipeline>
```

Language Constructs

- **`p:choose`**
- **`p:for-each`**
- **`p:viewport`**
- **`p:try/p:catch`**
- **`p:pipeline-library`**

p:choose

```
<p:choose name="testroot">
  <p:when test="/root[@version=2]">
    <p:output port="result"/>
    ...
  </p:when>
  <p:when test="/root">
    <p:output port="result"/>
    ...
  </p:when>
  <p:otherwise>
    <p:output port="result"/>
    ...
  </p:otherwise>
</p:choose>
```

p:for-each

```
<p:for-each>
  <p:iteration-source select="//chapter">
    <p:pipe step="x" port="y"/>
  </p:iteration-source>
  <p:output port="result"/>

  <p:xslt>
    <p:input port="stylesheet" href="docbook.xsl"/>
  </p:step>
</p:for-each>
```

p:viewport

```
<p:declare-step type="q:summarize"
       xmlns:q="...">
  <p:input port="entry"/>
  <p:output port="result"/>
</p:declare-step>

<p:viewport match="atom:entry">
  <p:viewport-source>
    <p:pipe step="generate-atom-feed" port="result"/>
  </p:viewport-source>
  <p:output port="result"/>

  <q:summarize/>
</p:viewport>
```

p:try

```
<p:try name="tryit">
  <p:group>
    <p:output port="out"/>
    ...
  </p:group>
  <p:catch>
    <p:output port="out"/>
    ...
      <p:input ...>
        <p:pipe step="tryit" port="errors"/>
      </p:input>
    ...
  </p:catch>
</p:try>
```

p:pipeline-library

```
<p:pipeline-library namespace="...">

  <p:import href="os-library.xml"/>

  <p:pipeline name="xinclude-and-style-with-docbook">...

  <p:pipeline name="xinclude-and-style">...

  <p:pipeline name="get-tide-information">...

</p:pipeline-library>
```

Options

- **Components can declare that they accept options.**
- **Steps can compute option values using XPath.**
- **Options are strings.**

Parameters

- **Any number of parameters can be passed to steps.**
- **Parameter names are not known in advance.**
- **Pipelines can manipulate sets of parameters for different steps.**

Q&A

- **We're making progress**
- **Last call “real soon now”**
- **On to the demo!**