

# **XProc: An XML Pipeline Language (Late Breaking)**

**Norman Walsh  
Sun Microsystems, Inc.**

# 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.**
- **Just finished our second face-to-face meeting last week.**
- **Our charter expires on 31 Oct 2007. *We will be finished.***

# 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?**

# What are we doing?

- **We're working on the former goal: developing a language for describing how XML components are joined together.**

# 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

Parse Decrypt SPARQL Query  
Encrypt Filter Sign  
Validate Load Absolutize  
Schematron XQuery Store Label  
Strip WS XSLT Tag Soup Exec  
Aggregate XSLT2 XInclude Wrap  
Render to... Prettyprint Delete  
Chunk httpRequest Rename  
Subsequence Soap Exchange  
RELAX NG Sort

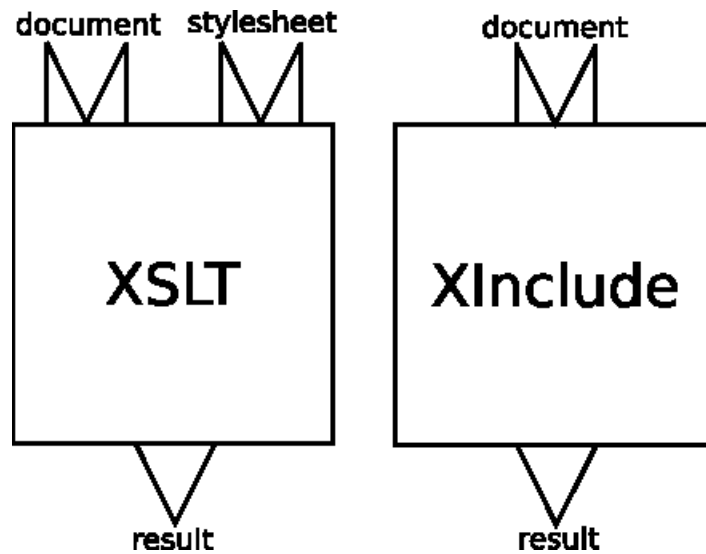
# Hasn't this been done already?

**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**

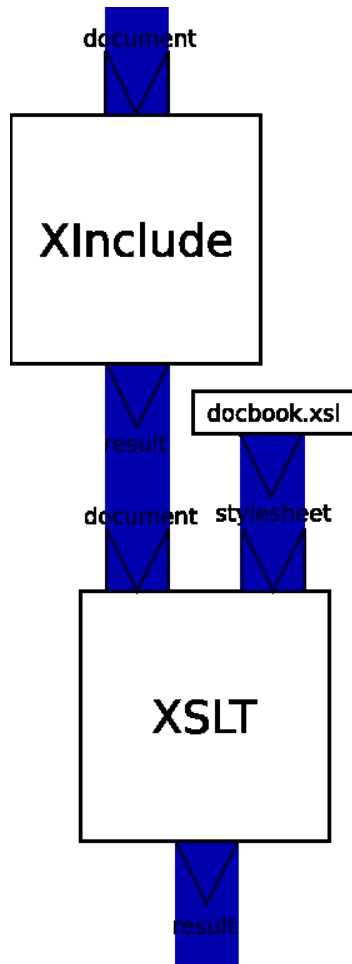
# Design Goals

- **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 Components



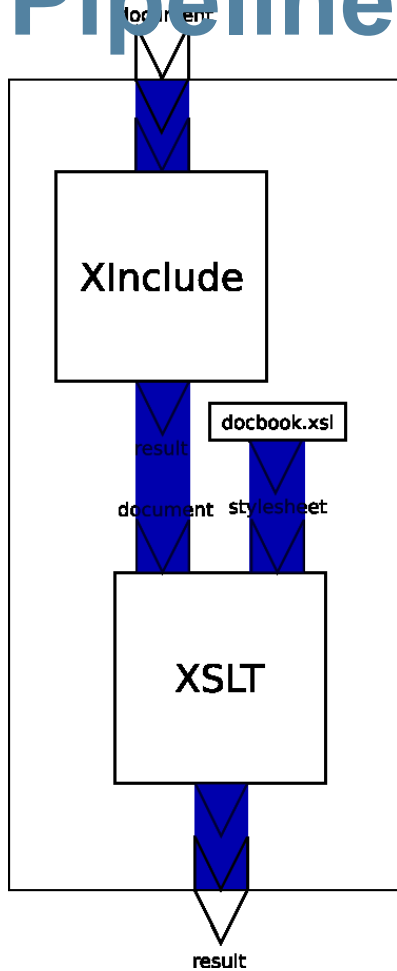
# The Pipeline Analogy



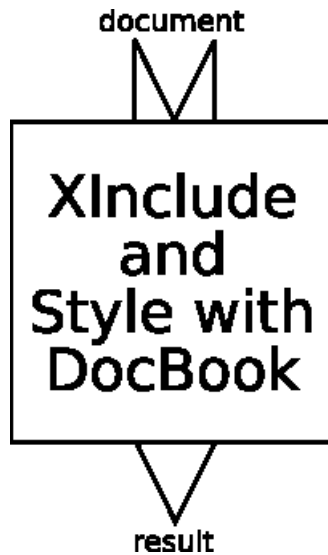
# What flows through pipes?

- **The “water” of our pipelines are XML documents:**
  - > **Not elements or nodes.**
  - > **Not XDMs or Infosets or PSVIs (or rather, any of those.)**

# Components can be Grouped into Pipelines



# Pipelines are Components



# Be Extreme!

**Show us the angle brackets! Ok, but...**

- **We're still making it up.**
- **We don't (all) like it.**
- **Some (or all) of it will change.**
- **I've *already* changed some of it!**

# A Little Terminology

- **Components have named “ports”.**
- **Ports are declared for components.**
- **Inputs and outputs bind document streams to ports.**
- **Steps instantiate components.**
- **Steps have a name and a “component kind”.**

# Declaring Components

```
<p:declare-component component="p:xinclude">  
  <p:declare-input port="document" sequence="yes"/>  
  <p:declare-output port="result" sequence="yes"/>  
</p:declare-component>
```

```
<p:declare-component component="p:xslt">  
  <p:declare-input port="document"/>  
  <p:declare-input port="stylesheet"/>  
  <p:declare-output port="result" sequence="yes"/>  
</p:declare-component>
```

# Using Components

```
<p:step name="exinc" component="p:xinclude">  
  <p:input port="document" href="doc.xml"/>  
</p:step>
```

```
<p:step name="xform" component="p:xslt">  
  <p:input port="document" source="exinc!result"/>  
  <p:input port="stylesheet" href="docbook.xsl"/>  
</p:step>
```

# Making a Pipeline

```
<p:pipeline>
  <p:declare-input port="document" />
  <p:declare-output port="result" source="xform!result" />

  <p:step name="exinc" component="p:xinclude">
    <p:input port="document" source="document" />
  </p:step>

  <p:step name="xform" component="p:xslt">
    <p:input port="document" source="exinc!result" />
    <p:input port="stylesheet" href="docbook.xsl" />
  </p:step>
</p:pipeline>
```

# Language Constructs

- `p:choose`
- `p:for-each`
- `p:viewport`
- `p:try`
- `p:pipeline-library`

## p:choose

```
<p:choose name="testroot" source="x!y">
  <p:when test="/root">
    <p:declare-output port="result" source="s1!result"
    ...
    <p:step name="s1" ...>
  </p:step>
</p:when>
<p:when test="/root" source="x!z">
  <p:declare-output port="result" source="s2!output"
  ...
  <p:step name="s2" ...>
  </p:step>
</p:when>
</p:choose>
```

## p:for-each

```
<p:for-each name="dochaps">
  <p:declare-input port="chaps" source="x!y" select="*" />
  <p:declare-output port="result" source="xform!result" />

  <p:step name="xform" component="p:xslt">
    <p:input port="document" source="!chaps" />
    <p:input port="stylesheet" href="docbook.xsl" />
  </p:step>
</p:for-each>
```

## p:viewport

```
<p:declare-component component="q:summarize"  
    xmlns:q="..."  
    q:class="com.example...">  
  <p:declare-input port="entry"/>  
  <p:declare-output port="result"/>  
</p:declare-component>
```

```
<p:viewport name="summarize">  
  <p:declare-input port="entry" source="x!y" select="...">  
  <p:declare-output port="result" source="summarize" select="...">  
  
  <p:step name="summarize" component="q:summarize">  
    <p:input port="entry" source="!entry"/>  
  </p:step>  
</p:viewport>
```

## p:try

```
<p:try name="tryit">
  <p:declare-output port="out"/>
  ...
  <p:catch>
    <p:declare-output port="out"/>
    ...
    <p:input source="tryit!err"/>
    ...
  </p:catch>
</p:try>
```

## p:pipeline-library

```
<p:pipeline-library>
```

```
  <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>
```

# Parameters

- **Components can declare that they accept parameters.**
- **Steps can bind values to parameters.**
- **Parameters are strings.**

# Q&A

- **We're making progress**
- **There are still lots of unanswered questions.**
- **Planned First Public Working Draft in a month or so.**
- **No demo today.**