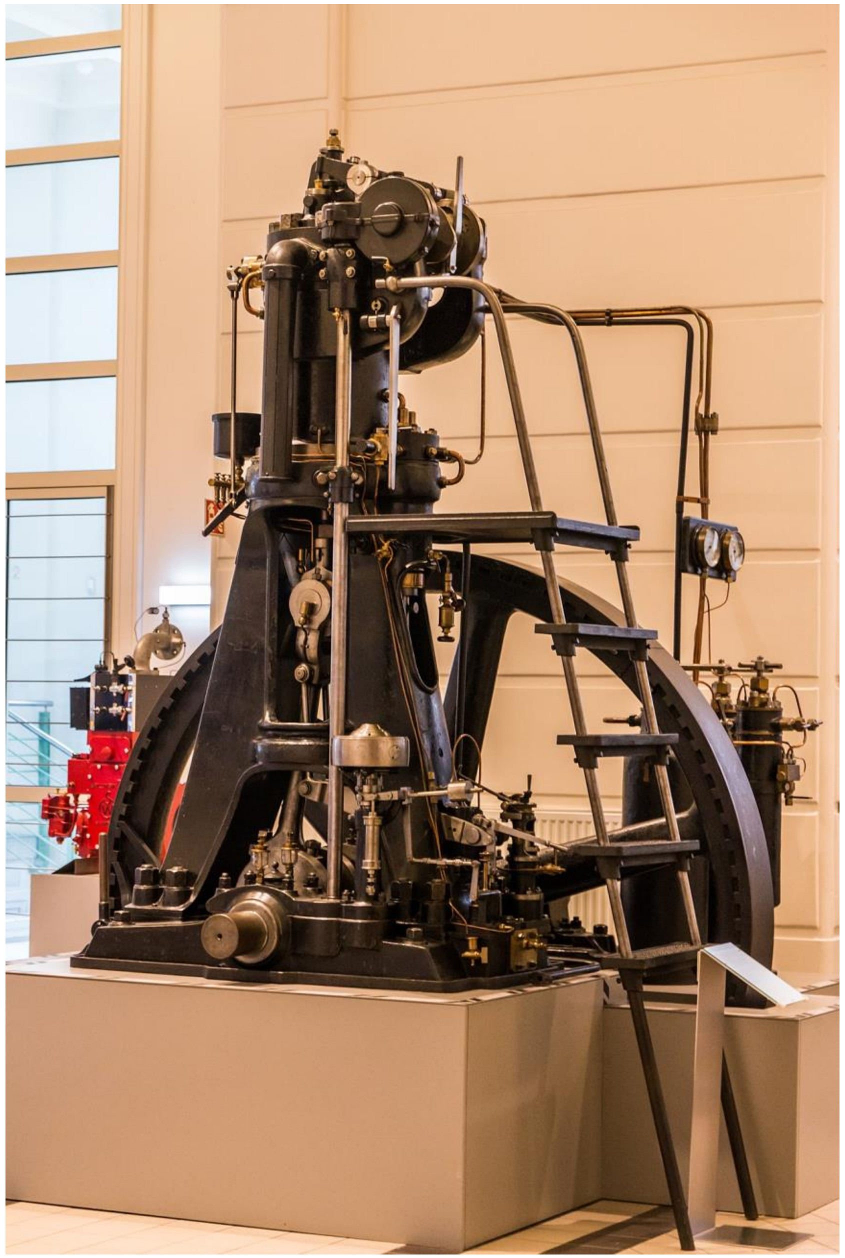

Diesel engine built by Langen & Wolf under licence, 1898
Image by Johannes Maximilian - Own work, GFDL 1.2,
https://commons.wikimedia.org/w/index.php?curid=70225625

DSL Engines running on XProc

Erik Siegel
erik@xatapult.nl
Declarative Amsterdam 2024
XATAPULT
CONTENT ENGINEERING

Who am I?

- Erik Siegel (Xatapult)
- Content Engineer, XML specialist, technical writer, trainer
- Technical background
- Fully specialized in XML and accompanying programming languages
- Member of the XProc editorial committee
- Author of three XML related books
- Contact details:

  erik@xatapult.nl
  www.xatapult.nl
  www.linkedin.com/in/esiegel/
  +31 6 53260792

Schematron
A Language for Validating XML
Erik Siegel
XProc 3.0
Programmer Reference
Erik Siegel
O'REILLY
eXist
A NOSQL DOCUMENT DATABASE AND APPLICATION PLATFORM
Erik Siegel & Adam Retter

DSL: Domain-Specific Languages

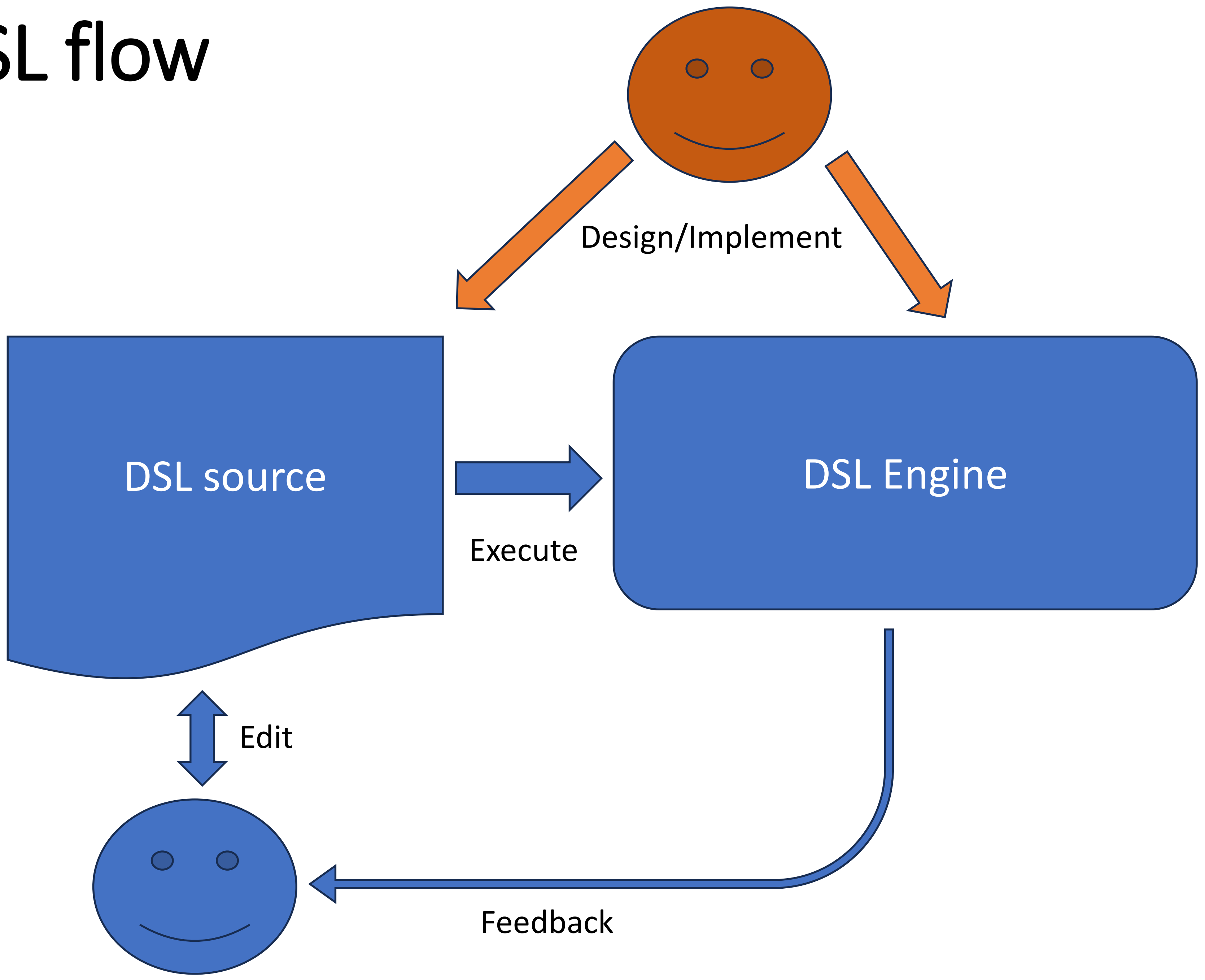
A domain-specific language (DSL) is a computer language specialized to a particular application domain. This is in contrast to a general-purpose language (GPL), which is broadly applicable across domains. (…)

Creating a domain-specific language (with software to support it), rather than reusing an existing language, can be worthwhile if the language allows a particular type of problem or solution to be expressed more clearly than an existing language would allow and the type of problem in question reappears sufficiently often. Pragmatically, a DSL may be specialized to a particular problem domain, a particular problem representation technique, a particular solution technique, or other aspects of a domain.

Source: Wikipedia

XATAPULT

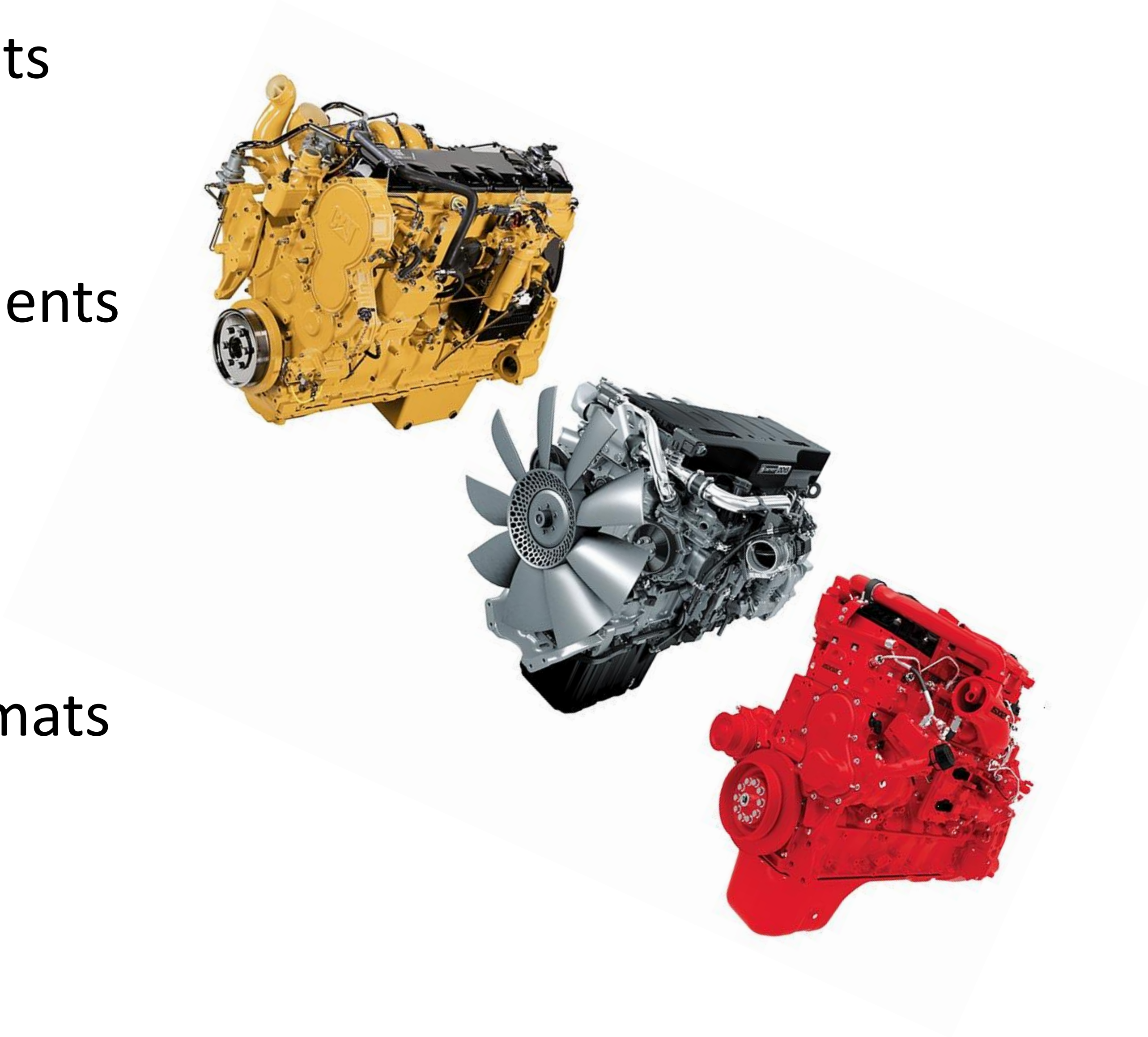
DSL flow



- Manipulate files and directories
- Transform documents
- Communicate with webservices
- Setup complicated structures
- …

What kind of DSLs?

- Creating/copying/moving documents
  - On disk
  - On the web
- Inspecting and transforming documents
  - Generate documentation
  - Publishing
- Task oriented processing
  - Like Ant (but not generic)
- Creating/adapting "XML in ZIP" formats
  - Office documents
  - JAR files
- Orchestrating webservices

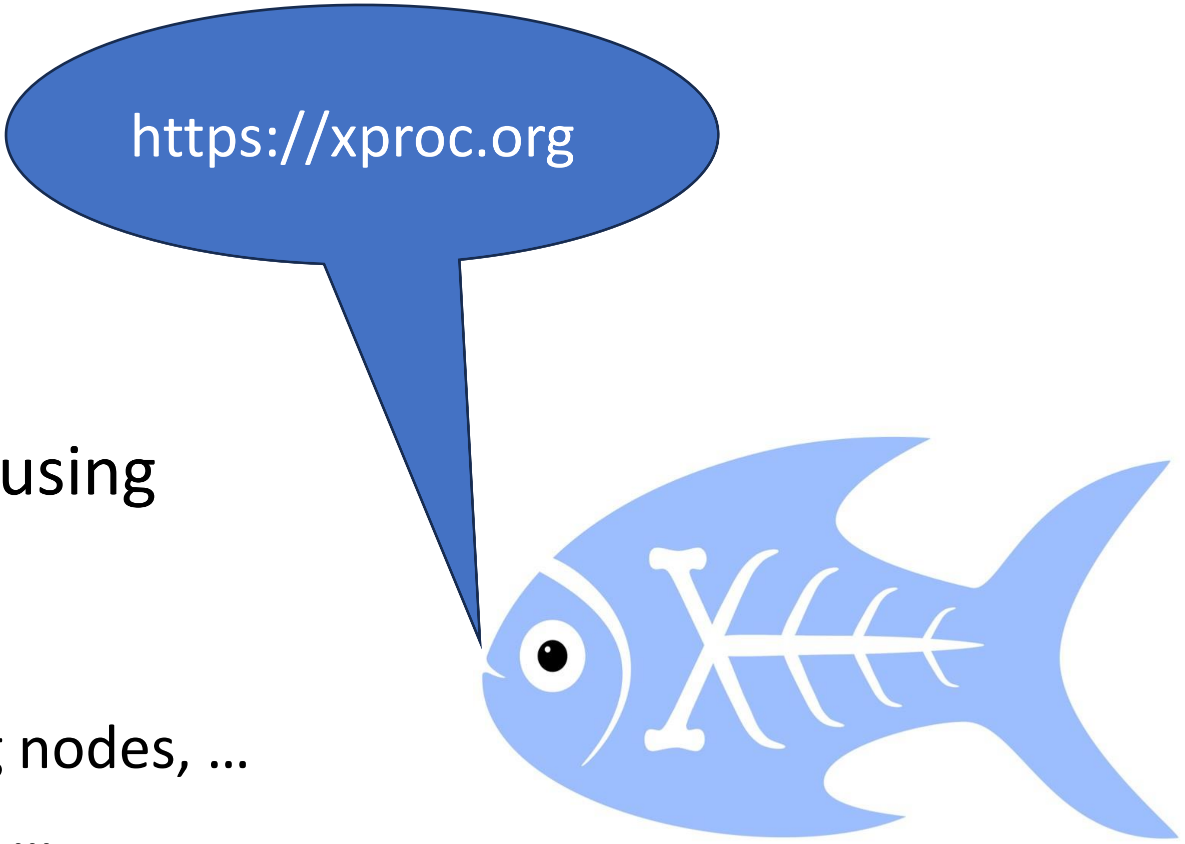


XML as base language for a DSL

- Well-understood syntax and semantics
- Designer: Define the DSL using schemas
- User: Edit and check with schema support
- Implementor: Many programming languages supporting XML

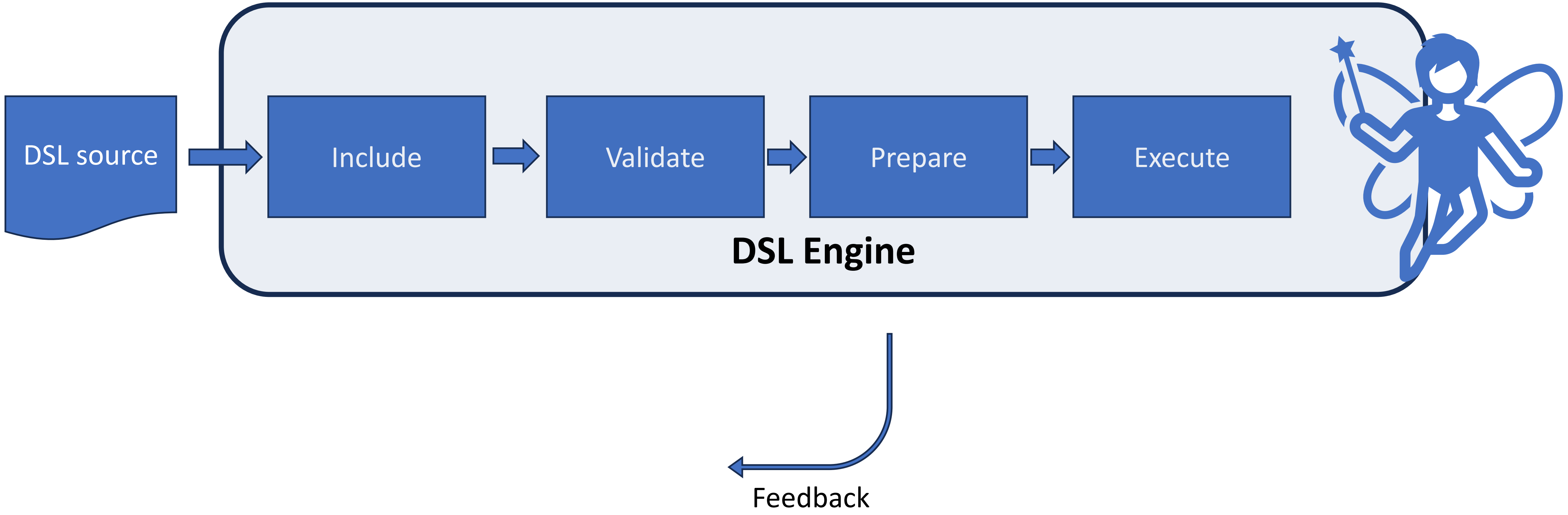
Examples of XML based DSLs: Ant, Maven config, Spring config, …

XProc for executing a DSL

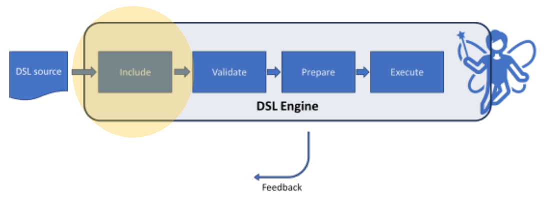


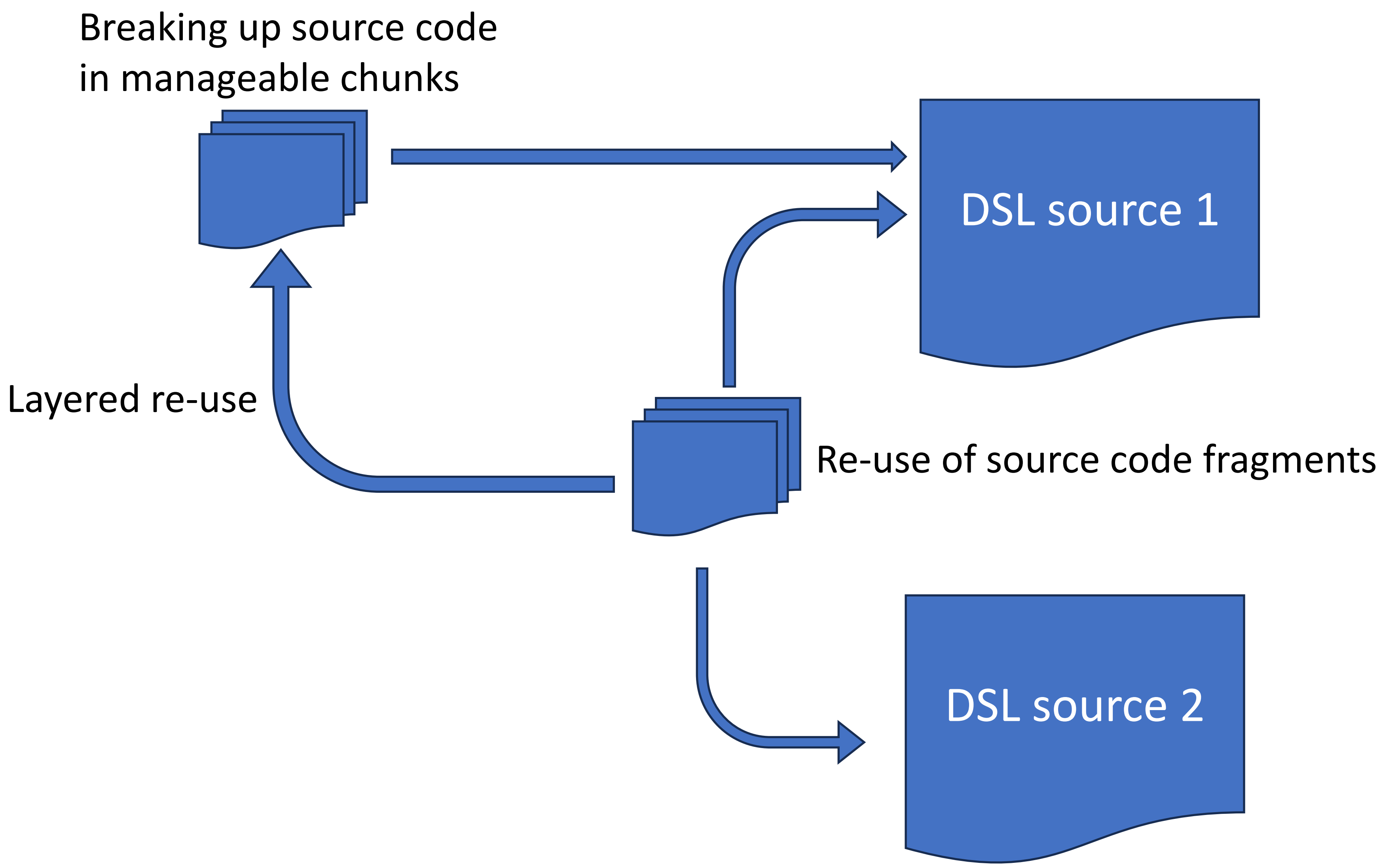
- XML based programming language
- Process XML (and other) documents using pipelines
- Pipelines consist of steps
  - Simple stuff: adding attributes, deleting nodes, …
  - Complex stuff: XSLT, XQuery, validation, …
  - Communicate with webservices
  - Manipulate files and directories
- Pipelines can loop, fork, merge, etc.
- Execution engine available (Morgana)

DSL processing using XProc



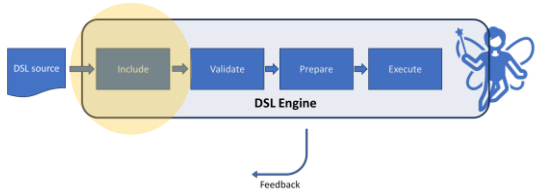
Adding modularity





Breaking up source code in manageable chunks
DSL source 1
Layered re-use
Re-use of source code fragments
DSL source 2

Adding modularity: includes



XProc supports the XInclude standard (https://www.w3.org/TR/xinclude/):

- In your source(s):

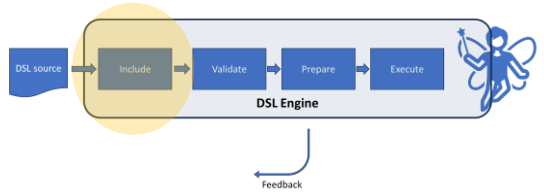
```
<xi:include href="…"/>
```

- In the XProc program:

```
<p:xinclude/>
<p:add-xml-base relative="false"/>
```

Records where the includes came from using xml:base attributes (here with absolute URIs)

Result of include handling



```
<main xmlns:xi="http://www.w3.org/2001/XInclude">
  …
  <xi:include href="includes/fragment1.xml"/>
  …
</main>
```

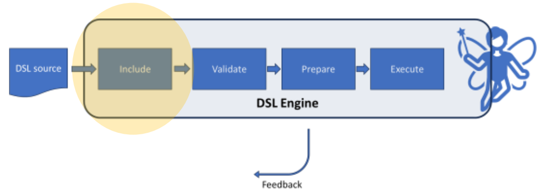
```
<command …>
  …
</command>
```

```
<p:xinclude/>
<p:add-xml:base relative="false"/>
```

```
<main xmlns:xi="http://www.w3.org/2001/XInclude"
      xml:base="file:/…/main.xml">
  …
  <command xml:base="file:/…/includes/fragment1.xml">
    …
  </command
  …
</main>
```

- Handy when debugging the compound document
- Needed for resolving relative URIs in the documents

Remove existing schema references

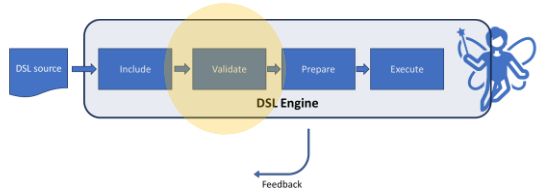


```
<?xml-model href="../sch/xprocref.sch" type="application/xml"
       schematypens="http://purl.oclc.org/dsdl/schematron"?>
<command xmlns="…" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="… ../xsd/xprocref.xsd">
  …
</command>
```

- References are usually relative ➔ no longer valid when document is written elsewhere
- Sometimes there are side effects of validation instructions

```
<p:delete match="@xsi:*"/>
<p:namespace-delete prefixes="xsi"/>
<p:delete match="processing-instruction(xml-model)"/>
```

Validation: avoid garbage in!



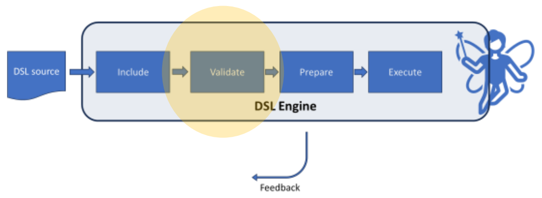
After handling includes, validate!

```
<p:validate-with-xml-schema>
  <p:with-input port="schema" href="…"/>
</p:validate-with-xml-schema>
```

Error message is a hard to interpret XML document…

```
<c:errors xmlns:c="http://www.w3.org/ns/xproc-step">
  <c:error code="err:XC0156" name="!1.1.1.2.1" type="p:validate-with-xml-schema"
    href="file:/C:/Data/Erik/work/xatapult/xtpxlib-common/xpl3mod/validate/validate.xpl" line="67" column="85" xmlns:p="http://www.w3.org/ns/xproc"
    xmlns:err="http://www.w3.org/ns/xproc-error">
    <report xmlns="http://www.xproc.org/ns/xvrl">
      <metadata>
        <timestamp>2024-09-26T09:19:42.04+02:00</timestamp>
        <document href="file:/C:/Data/Erik/work/xatapult/xprocref/src/xprocref.src.main.xml"/>
        <schema href="file:///C:/Data/Erik/work/xatapult/xprocref/xsd/xprocref.xsd" schematypens="http://www.w3.org/2001/XMLSchema"/>
        <validator name="org.apache.xerces.jaxp.validation.XMLSchemaFactory"/>
      </metadata>
      <detection severity="error">
        <location line="4616"
          xpath="/Q{http://www.xtpxlib.nl/ns/xprocref}xprocref[1]/Q{http://www.xtpxlib.nl/ns/xprocref}steps[1]/Q{http://www.xtpxlib.nl/ns/xprocref}step-
group[1]/Q{http://www.xtpxlib.nl/ns/xprocref}step[17]/Q{http://www.xtpxlib.nl/ns/xprocref}invalid[1]"/>
        <message>cvc-complex-type.2.4.a: Invalid content was found starting with element '{"http://www.xtpxlib.nl/ns/xprocref":invalid}'. One of
          '{"http://www.xtpxlib.nl/ns/xprocref":macrodefs, "http://www.xtpxlib.nl/ns/xprocref":signature}' is expected.</message>
      </detection>
      <digest/>
    </report>
  </c:error>
</c:errors>
```

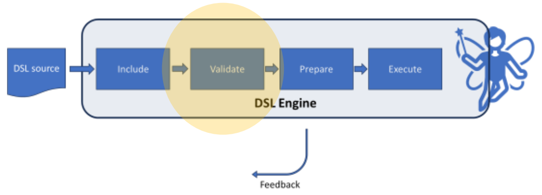
Simplify the validation error message



```
<p:try>

  <p:validate-with-schema>
    <p:with-input port="schema" href="..."/>
  </p:validate-with-xml-schema>

  <p:catch>

    <!-- Re-raise the error with simplified message -->
    <p:error code="schema-validate-error">
      <p:with-input>
        <p:inline content-type="text/plain">{string(/*/c:error[1])}</p:inline>
      </p:with-input>
    </p:error>

  </p:catch>
</p:try>
```

Only the *first* actual error message

Add a Schematron-schema

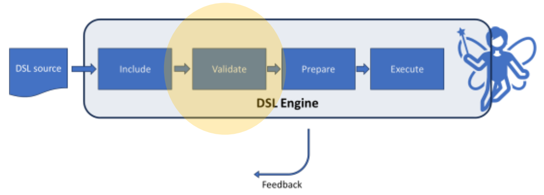


- After schema validation, validate using Schematron:

```
<p:validate-with-schematron>
  <p:with-input port="schema" href="…"/>
</p:validate-with-schematron>
```

- Checking things like:
  - Identifier references (across includes)
  - …

Simplify the Schematron error message

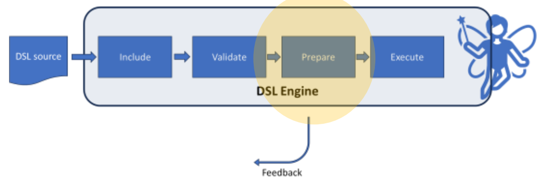


Error XML contains the full SVRL output of the Schematron validation...

```
<p:try>

  <p:validate-with-schematron >
    <p:with-input port="schema" href="..."/>
  </p:validate-with-schematron>

  <p:catch>

    <p:error code="schematron-validate-error">
      <p:with-input>
        <p:inline content-type="text/plain">{string(/*//svrl:failed-assert[1])}</p:inline>
      </p:with-input>
    </p:error>

  </p:catch>
</p:try>
```

Turn the first SVRL failed assert error message into text

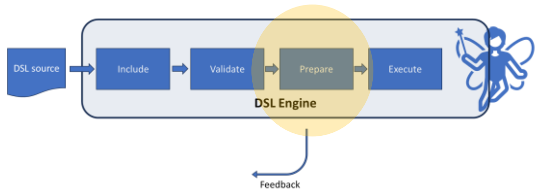
Prepare your DSL XML for execution



- Processing a DSL usually means: accessing external resources
    - Files to read or write
    - Webservices to call
- Paths/URIs are often implicit (*because* it is a DSL…)
- Compute all information for the basic actions up-front
    - Record in additional attributes (or elements)
- Not a task for XProc itself ➔ Use XSLT

```
<p:xslt>
  <p:with-input port="stylesheet" href="prepare-dsl.xsl"/>
</p:xslt>
```

Prepare your DSL XML - Example



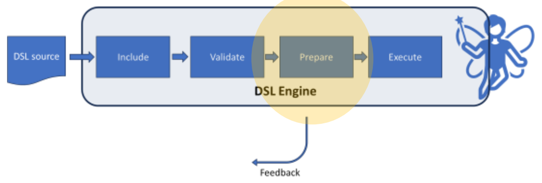
```
<commands>
  ...
  <add-special-document/>
  ...
</commands>
```

XSLT

```
<commands>
  ...
  <add-special-document
    _href-source="file:///C:/source/path/special.xml"
    _href-target="file:///C:/target/path/special.xml"
  />
  ...
</commands>
```

```
<p:for-each>
  <p:with-input select="//add-special-document"/>
  <p:file-copy href="{/*/@_href-source}" target="{/*/@_href-target}"/>
</p:for-each>
```

Prepare your DSL XML – Handy XProc step



There is a handy XProc step for the preparations for making existing attributes absolute: **`p:make-absolute-uris`**

```
<commands>
  ...
  <add-special-document href="abc.xml"/>
  ...
</commands>
```

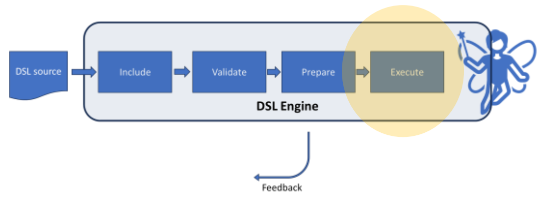
XProc

```
…
<p:make-absolute-uris
   match="add-special-document/@href"
   base-uri="file:///some/path/"/>
…
```

```
<commands>
  ...
  <add-special-document href="file:///some/path/abc.xml"/>
  ...
</commands>
```

XATAPULT

Execute your DSL XML



Use **`p:viewport`** to iterate over the things to do

```
…
  <action>
    …
  </action>
…
```

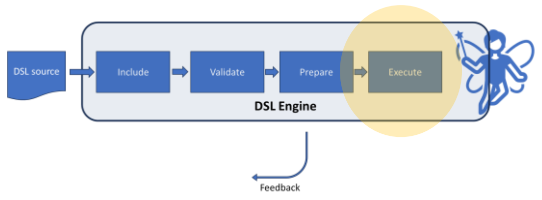
A viewport only sees the matched element

```
<p:viewport match="action">

      … more XProc code


</p:viewport>
```

All changes made in the viewport are re-inserted in the original document

This allows you to leave, for instance, status information about what happened

Execute your DSL XML



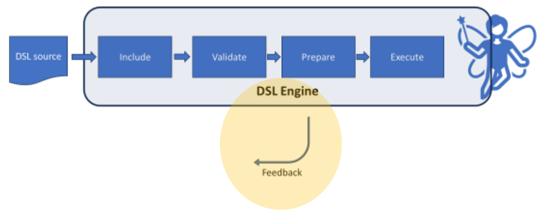
```
<commands>
  (prolog)
  <actions>
    <this .../>
    <that .../>
    ...
  </actions>
</commands>
```

```
<p:viewport match="/commands/actions/*">
  <p:choose>
    <p:when test="/*/self::this">
      ...
    </p:when>
    <p:when test="/*/self::that">
      ...
    </p:when>
    <p:otherwise>
      ... (raise error? ignore?)
    </p:otherwise>
  </p:choose>
</p:viewport>
```

Matched element becomes the document root element inside the viewport

XProc has no templating mechanism like XSLT… So, ugly dispatching code

Feedback from the DSL execution 1/3: Console message



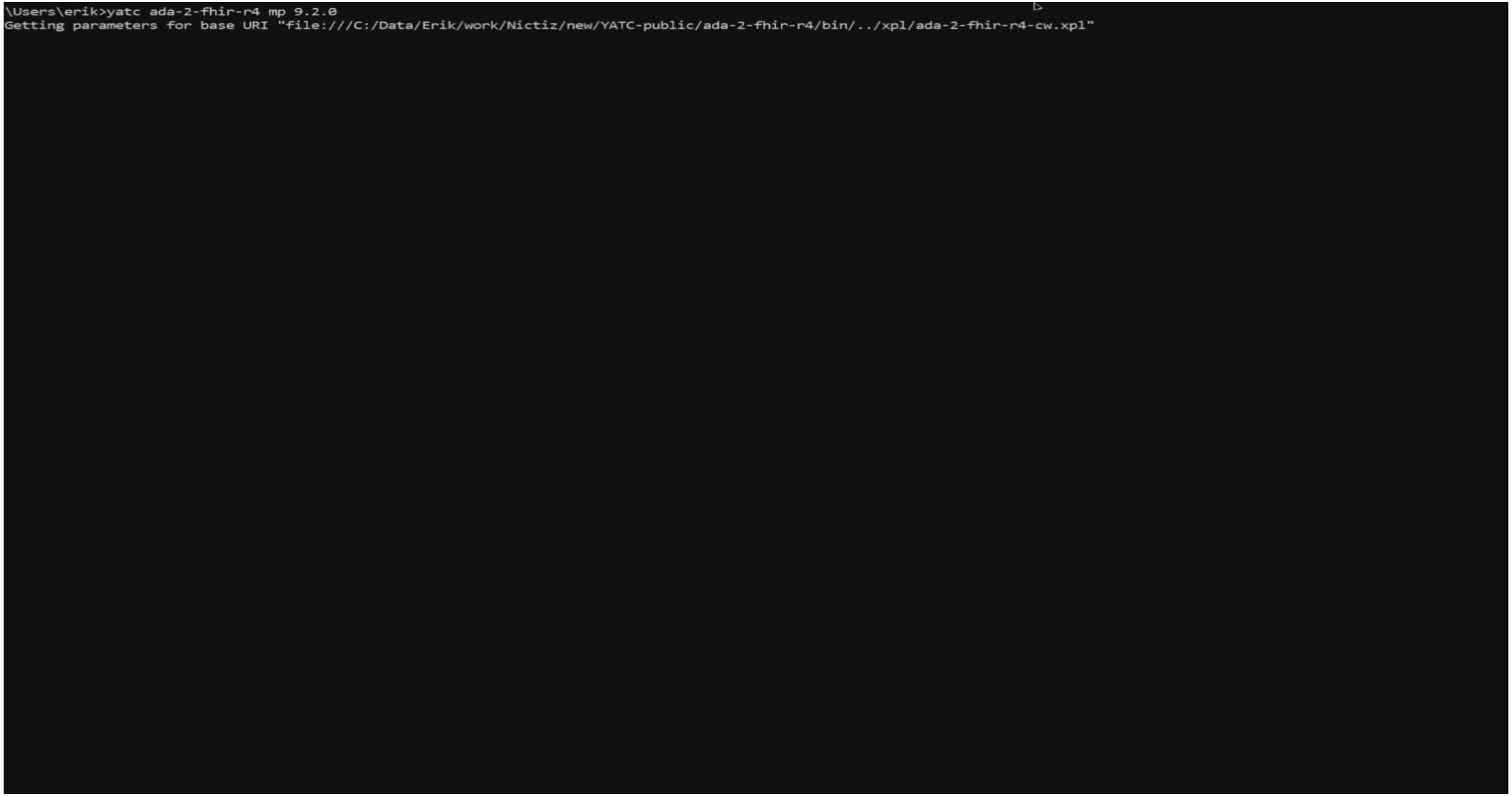
- XProc has a standard **`message`** attribute

```
<p:xslt message="Preparing…">
  …
</p:xslt>
```

- Or use empty **`p:identity`** steps for explicit messaging:

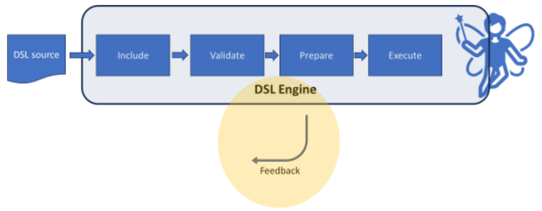
```
<p:identity message="We started processing!"/>
<p:identity message="- Input document {$href-input}"/>
<p:identity message="- Processing type {$processing-type}"/>
```

- Or use **`<xsl:message>`** instructions in an XSLT



```
\Users\erik>yatc ada-2-fhir-r4 mp 9.2.0
Getting parameters for base URI "file:///C:/Data/Erik/work/Nictiz/new/YATC-public/ada-2-fhir-r4/bin/../xpl/ada-2-fhir-r4-cw.xpl"
```

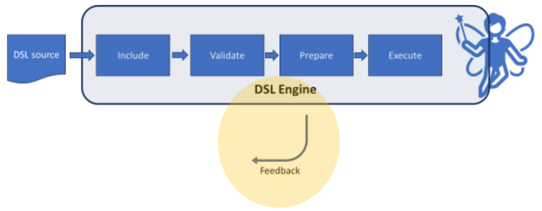
Feedback from the DSL execution 2/3: Additional documents



- Write stuff to disk
  - Intermediate results
  - Lists of actions taken
  - Overview of implicit values (directory names, etc.)
- Use the **`p:store`** step for this

```
… produce the document to write …
<p:store href="{$href-additional-dir}/interesting-info.xml/>
```

Feedback from the DSL execution 3/3: Final results

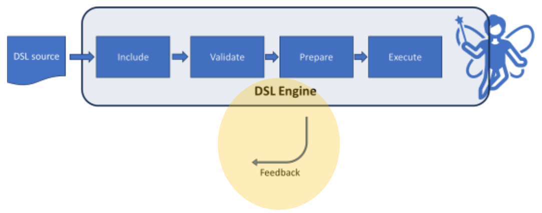


- An XProc pipeline usually has a primary **`result`** output port

```
<p:output port="result" primary="true" …/>
```

- What comes out is dumped on the console after processing
- Usually not necessary for the functionality of your DSL
- What comes out of this is under your control
- Use it to show important information:
  - Status (success/failure, etc.)
  - Statistics
  - Duration

How to compute and show the duration of a pipeline



```
<p:declare-step xmlns:p="http://www.w3.org/ns/xproc" version="3.0">

  <p:output port="result" primary="true"/>

  <p:variable name="start" as="xs:dateTime" select="current-dateTime()"/>

  ...

  <p:identity>
    <p:with-input>
      <dsl-execute status="{$status}" duration="{current-dateTime() - $start}"/>
    </p:with-input>
  </p:identity>

</p:declare-step>
```

```
<dsl-execute status="ok" duration="PT1M23S"/>
```

Wrap-up

- XML as base format for Domain Specific Languages
- XProc as language for the DSL processing
- Processing phases:
  - Include
  - Validate
  - Prepare
  - Execute

Help needed with XProc steps? https://xprocref.org/

DSL Engines running on XProc

XProc: https://xproc.org/

Erik Siegel

erik@xatapult.nl

Declarative Amsterdam 2024

XATAPULT

CONTENT ENGINEERING