XSLWeb: XSLT- and XQuery-only pipelines for the web

Maarten Kroon, Pieter Masereeuw

What does a webserver really do?

What does a webserver really do?

```
GET /demojam HTTP/1.1
Host: www.xmlprague.cz
User-Agent: Lynx/2.8.9dev.16 libwww-FM/2.14 SSL-MM/1.4.1 GNUTLS/3.5.17
```


What does a webserver really do?

```
GET /demojam HTTP/1.1
Host: www.xmlprague.cz
User-Agent: Lynx/2.8.9dev.16 libwww-FM/2.14 SSL-MM/1.4.1 GNUTLS/3.5.17
```

What does a webserver really do?

```
GET /demojam HTTP/1.1
Host: www.xmlprague.cz
User-Agent: Lynx/2.8.9dev.16 libwww-FM/2.14 SSL-MM/1.4.1 GNUTLS/3.5.17
```

What does a webserver really do?

```
<req:request>
        <req:path>/demojam</req:path>
        <req:request-URI>http://www.xmlprague.cz/demojam</req:request-URI>
</req:request>
```

What does a webserver really do?

```
<req:request>
        <req:path>/demojam</req:path>
        <req:request-URI>http://www.xmlprague.cz/demojam</req:request-URI>
</req:request>
```

XProc? / Piperack?
Apache Cocoon?
Servlex?
… ?

What does a webserver really do?

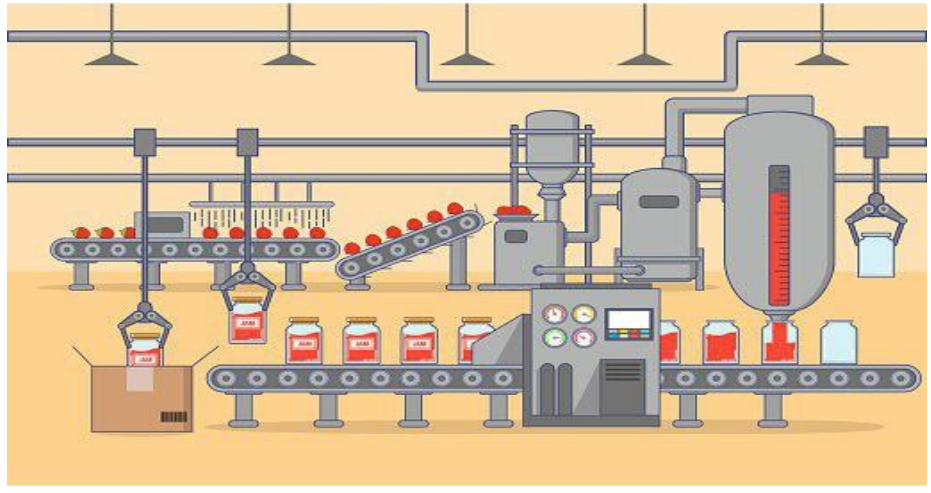
```
<req:request>
        <req:path>/demojam</req:path>
        <req:request-URI>http://www.xmlprague.cz/demojam</req:request-URI>
</req:request>
```

XSLWeb

XSLWeb

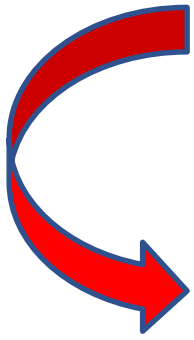
```
<req:request>
        <req:path>/demojam</req:path>
        <req:request-URI>http://www.xmlprague.cz/demojam</req:request-URI>
</req:request>
```

XSLWeb

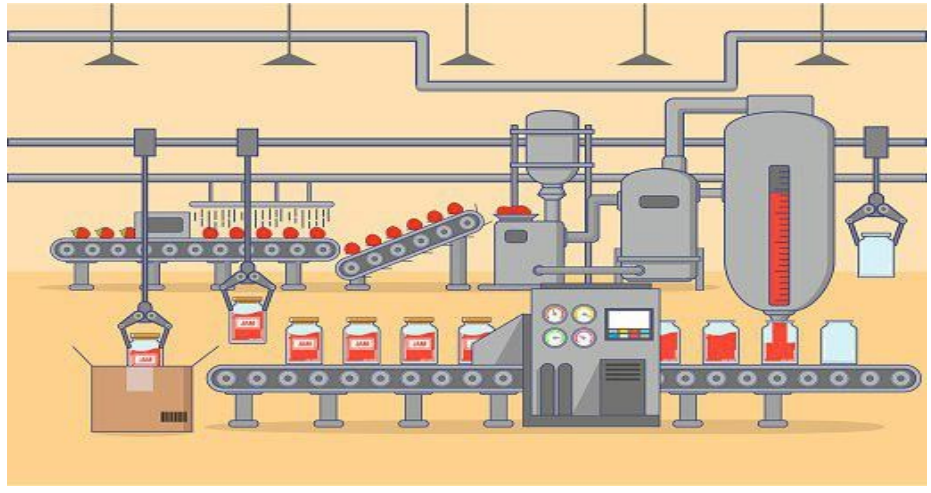


Pipelines in XSLWeb

```
<req:request>
        <req:path>/demojam</req:path>
        <req:request-URI>http://www.xmlprague.cz/demojam</req:request-
</req:request>
```



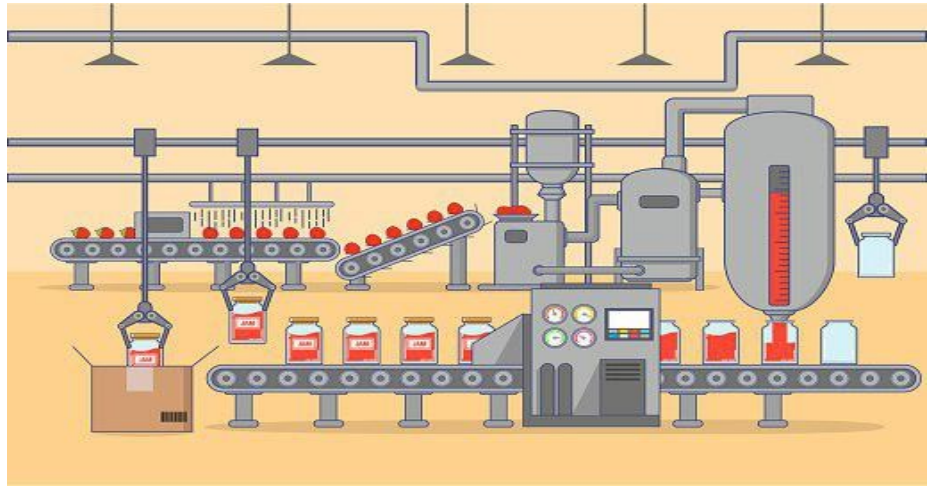
XSLWeb



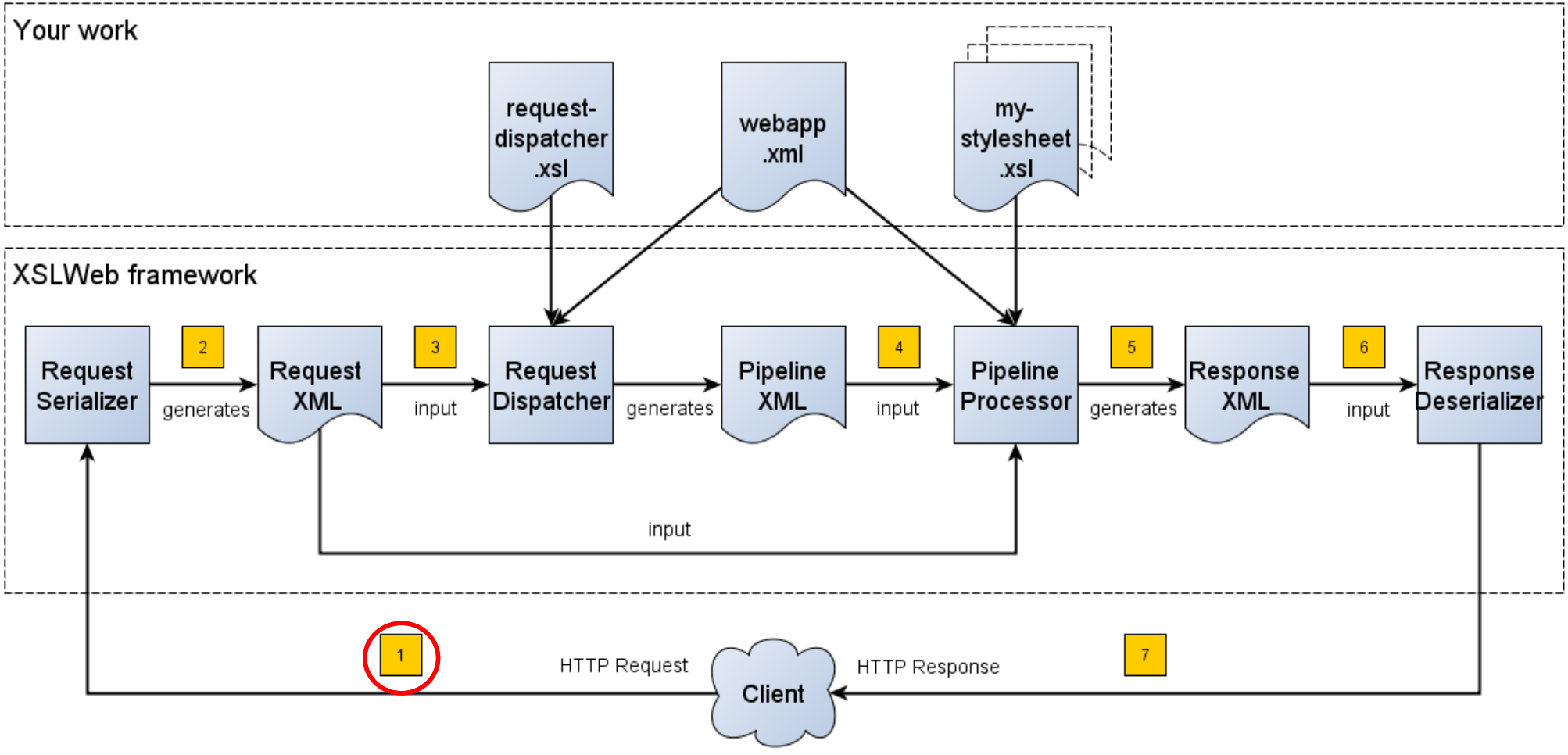
Pipelines in XSLWeb

```
<req:request>
        <req:path>/demojam</req:path>
        <req:request-URI>http://www.xmlprague.cz/demojam</req:request-
</req:request>
```

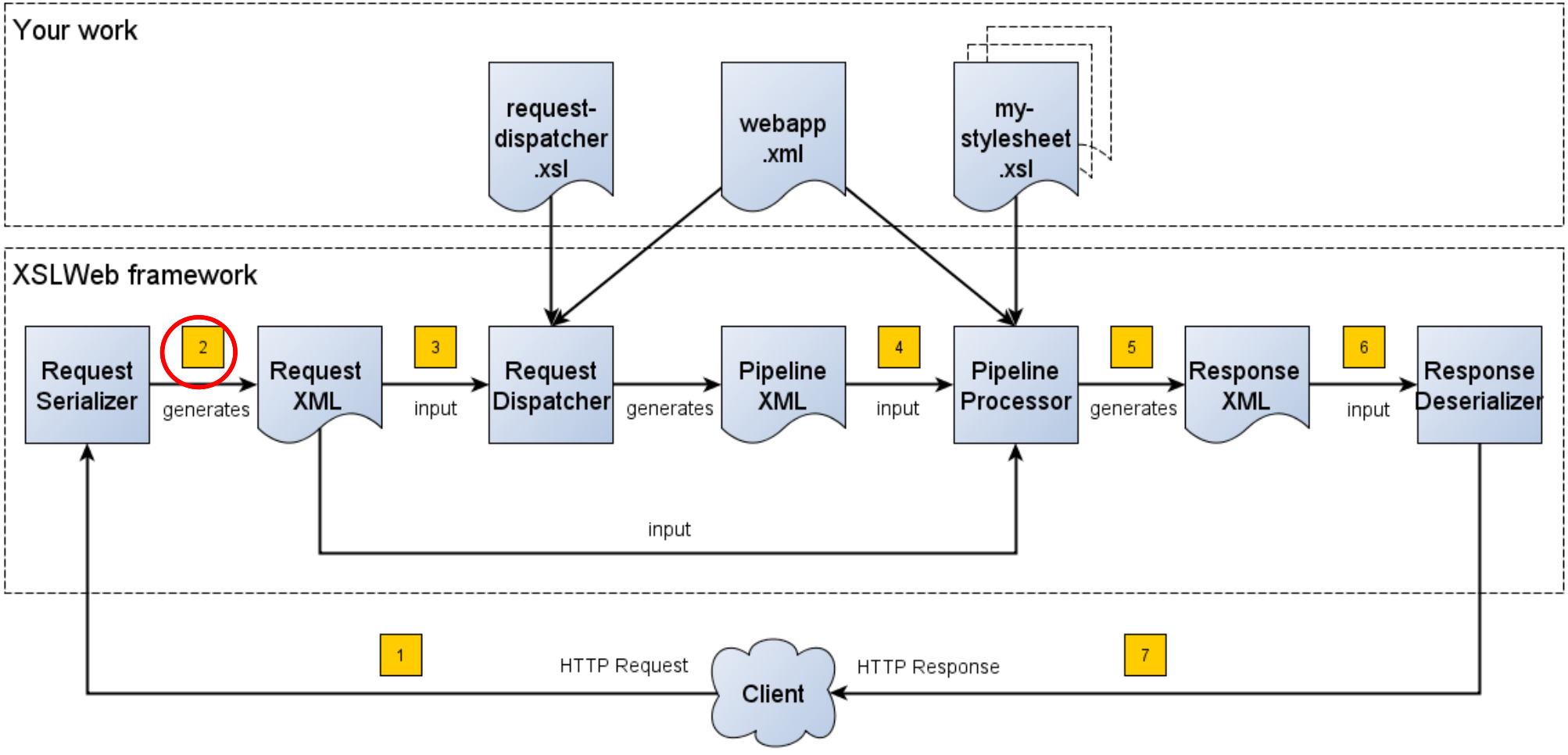
XSLWeb



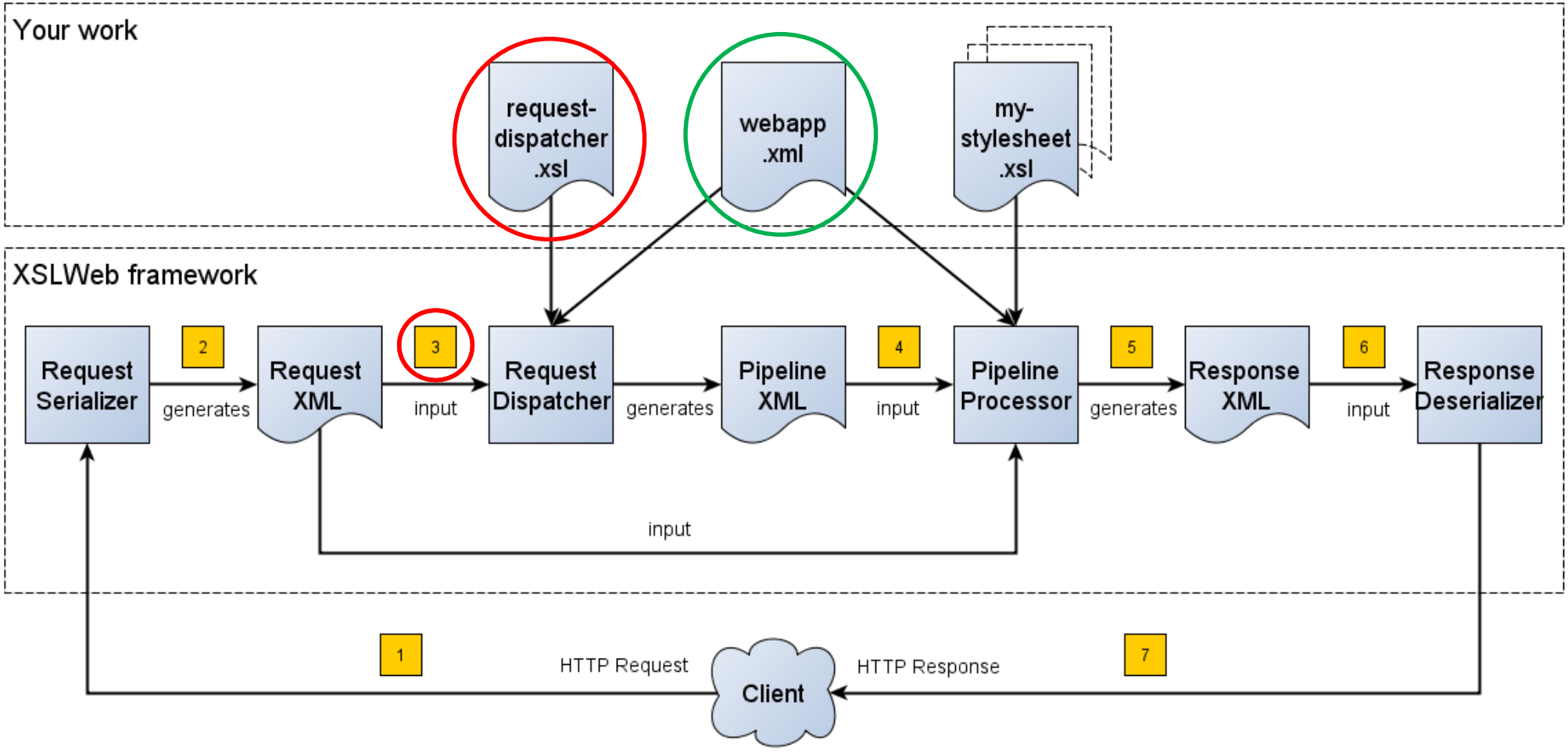
Pipelines in XSLWeb



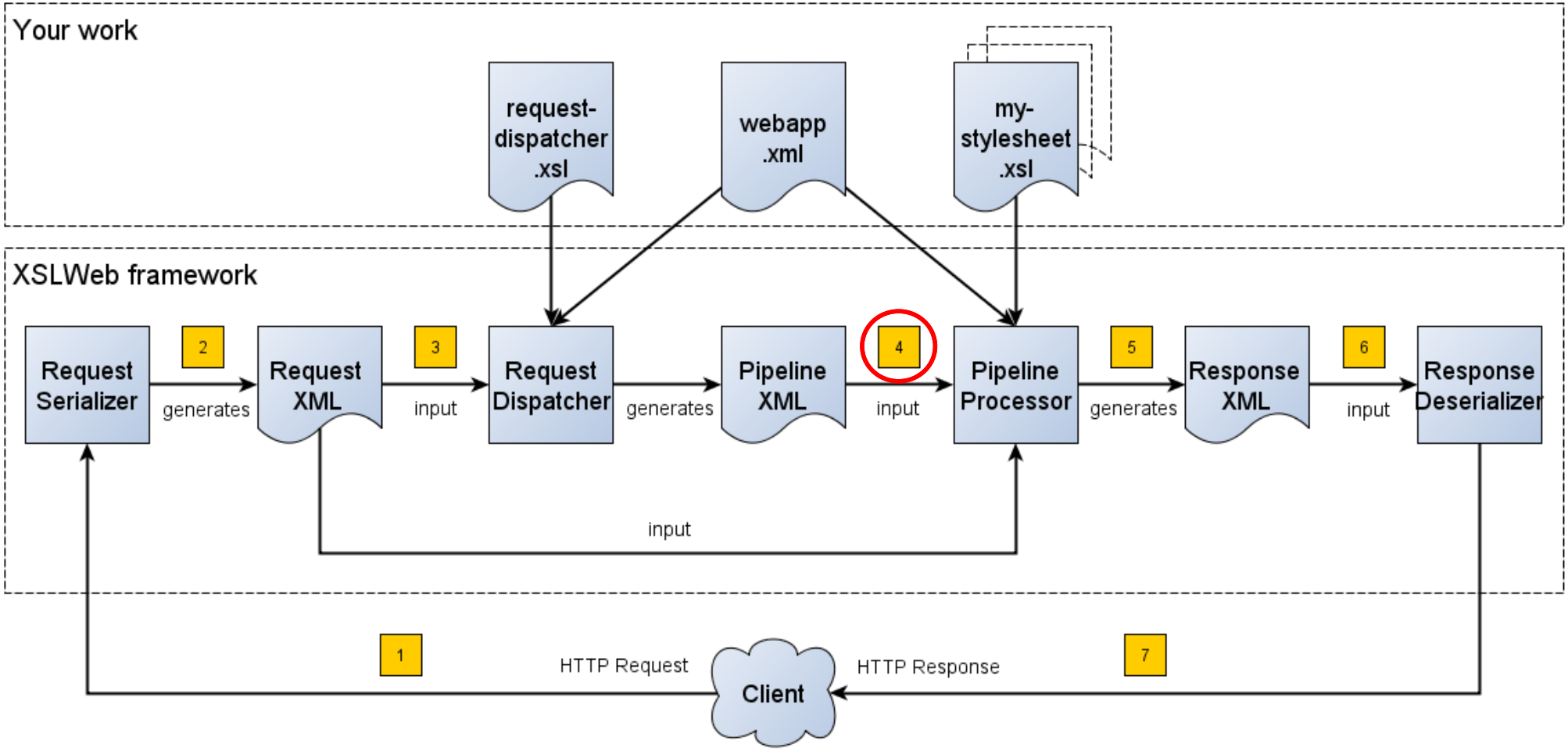
Pipelines in XSLWeb



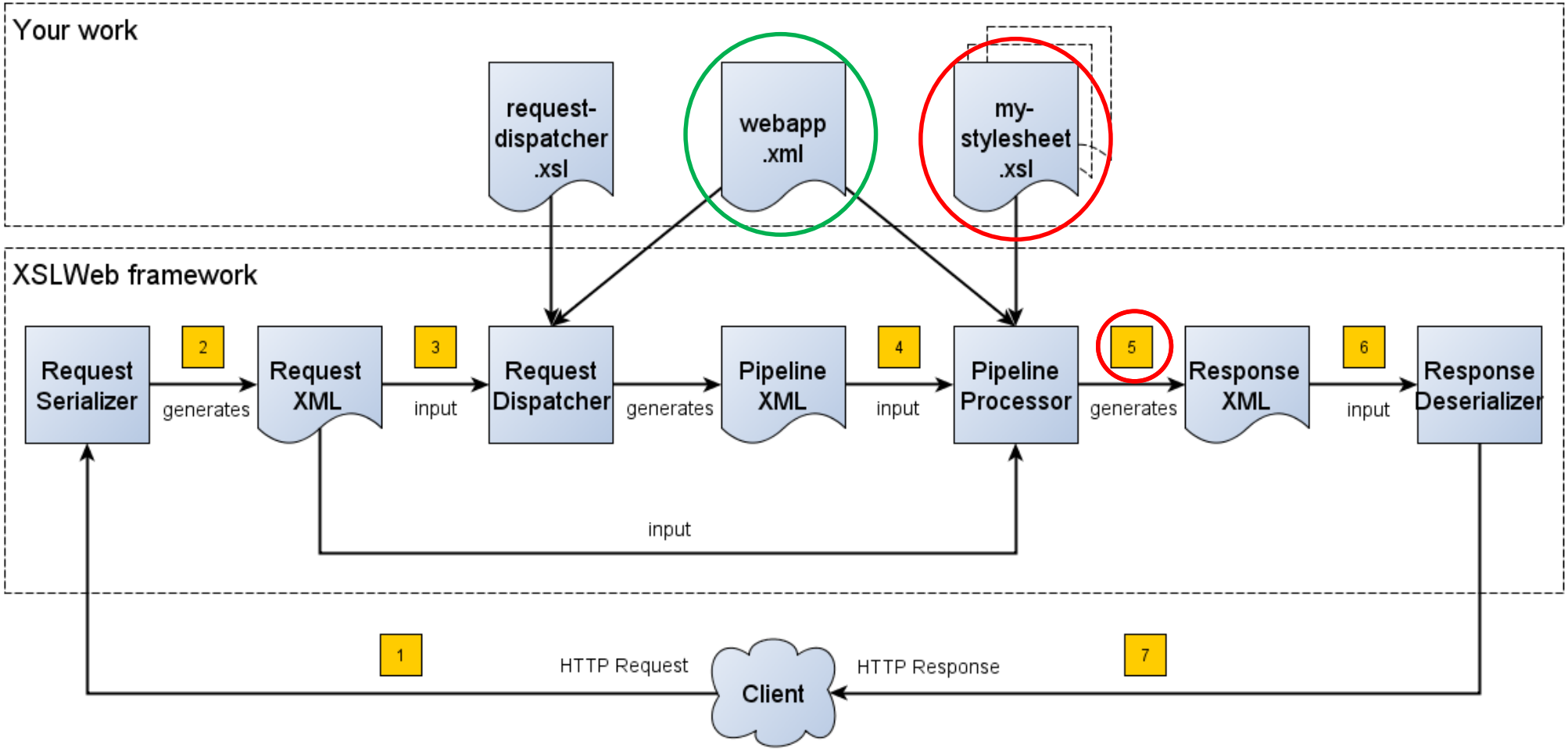
Pipelines in XSLWeb



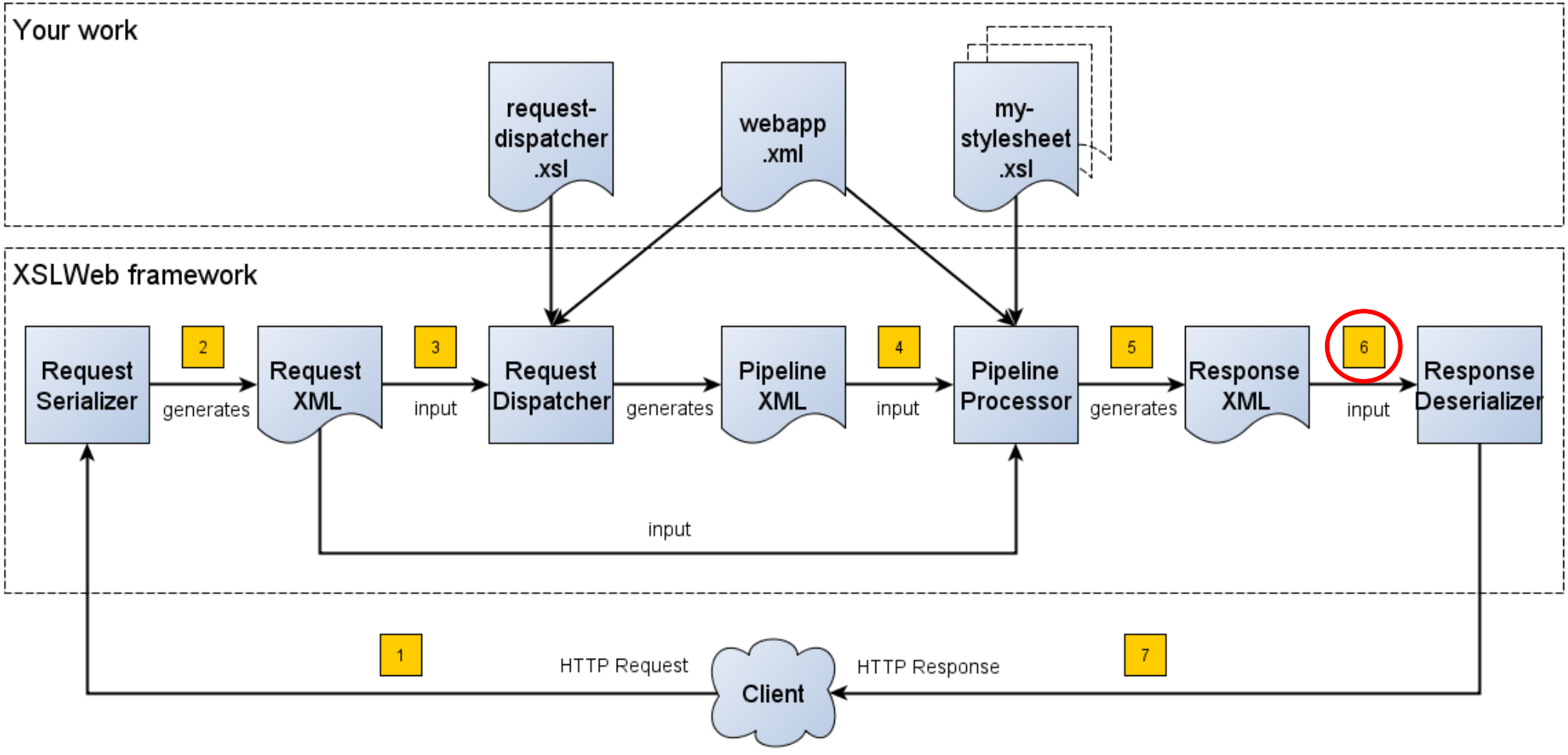
Pipelines in XSLWeb



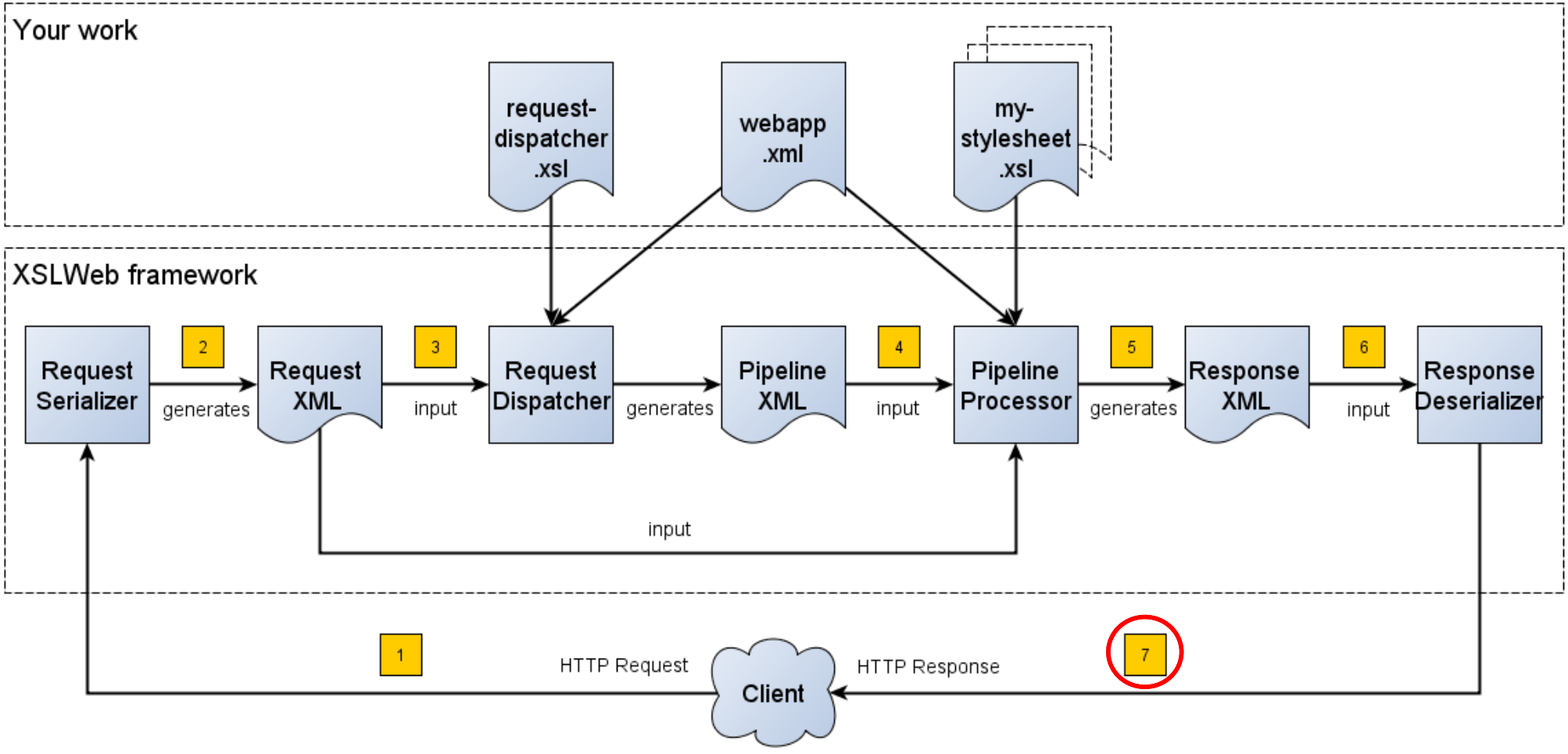
Pipelines in XSLWeb



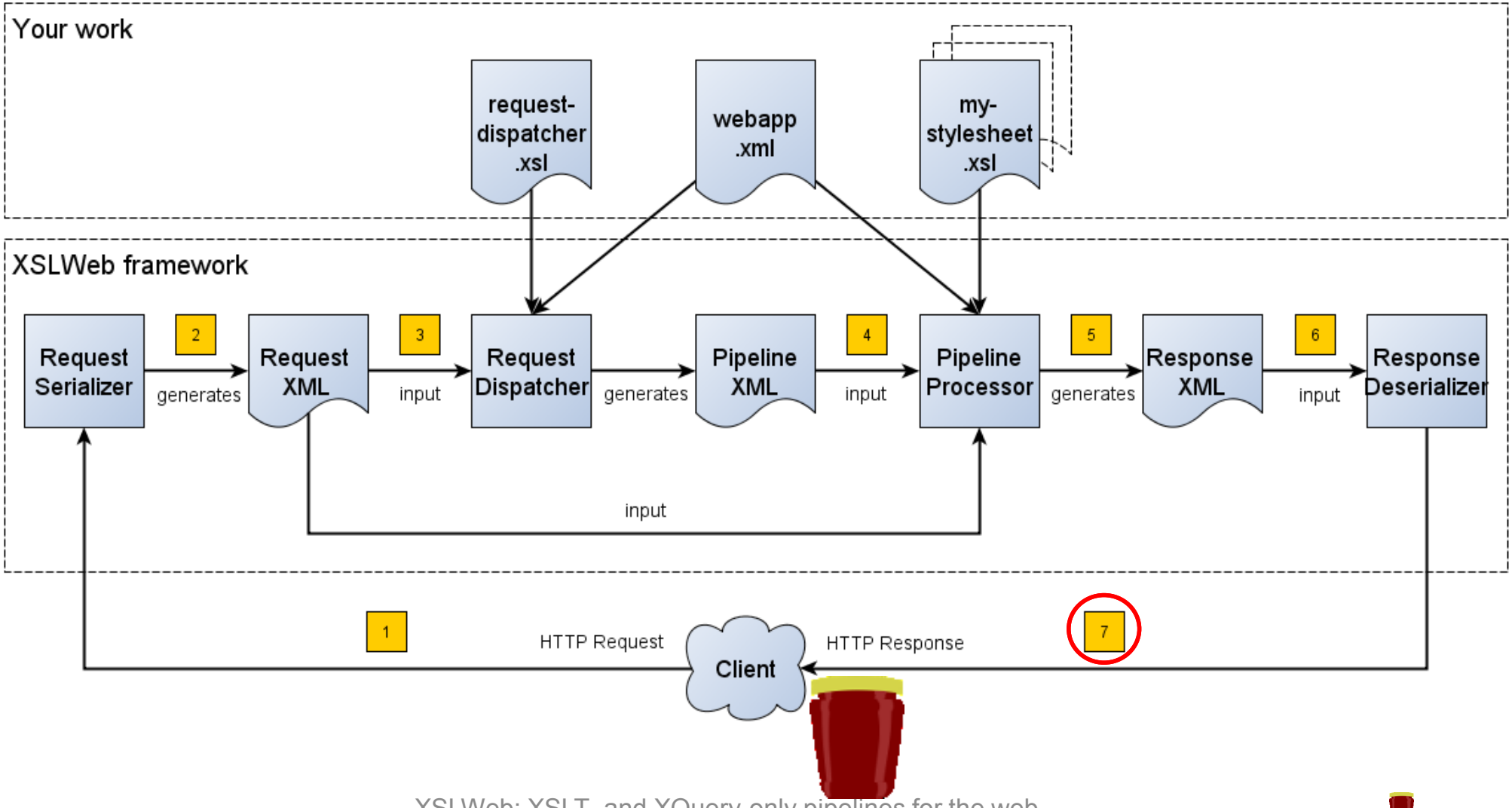
Pipelines in XSLWeb



Pipelines in XSLWeb

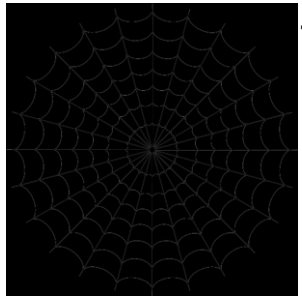


Pipelines in XSLWeb



XSLWeb: overview

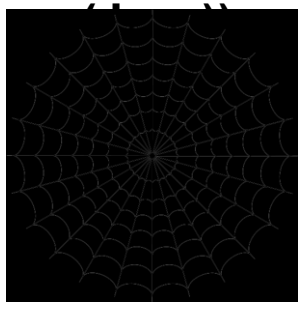
- XSLWeb is just a Java servlet
- Internally uses Saxon's efficient tree model (TinyTree)
- It gives access to the full HTTP request in an XML representation
- It offers an XML representation of the HTTP response
- It supports the full HTTP specification - GET, POST, PUT, and all other methods
- It makes pipelining trivially easy
- It allows XSLT and XQuery programmers to program things at the moment

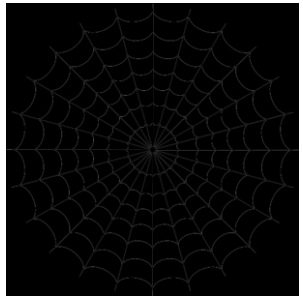


t - i.e. in their stylesheet or XQuery script

XSLWeb: overview

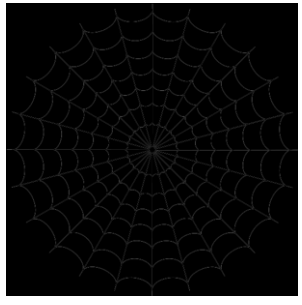
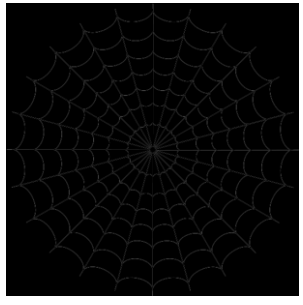
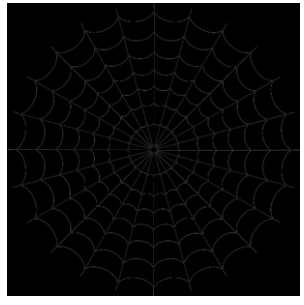
- It allows caching
- It allows access to static content (assets)
- It offers several extension functions to make web programming easy
- It allows the addition of user defined extension functions in Java, using the Saxon API.
- It is **in extensive daily use** by Dutch government organisations for web services and web sites (such as laws, treaties, patents and land registry).
- **But most importantly: nearly anything can be done in XSLT or Xquery (exceptions: the configuration file (XML) and extension functions**





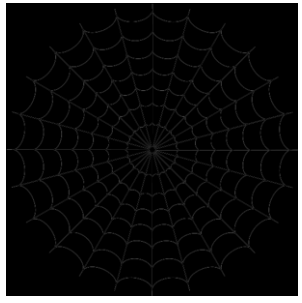
XSLWeb: extension functions

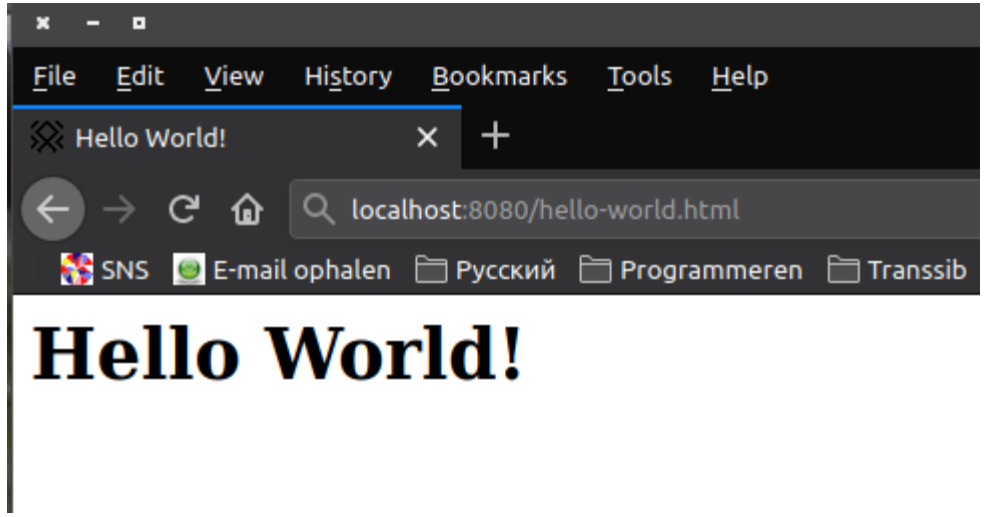
- Functions for manipulating the request, the session and the response
- EXPath file functions (Christian Grün, expath.org; own implementation)
- EXPath http functions (thanks to Florent Georges)
- Spawning external processes
- Sending e-mails
- Image processing
- ZIP file processing
- SQL processing (own implementation)

XSLWeb: the request dispatcher stylesheet

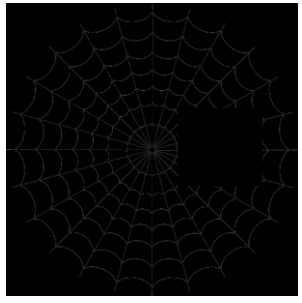
```
<xsl:stylesheet ..>

  <xsl:template match="/req:request[req:path eq '/hello-world.html']">
    <pipeline:pipeline>
      <pipeline:transformer name="hello-world"
           xsl-path="hello-world.xsl" log="true"/>
    </pipeline:pipeline>
  </xsl:template>

</xsl:stylesheet>
```

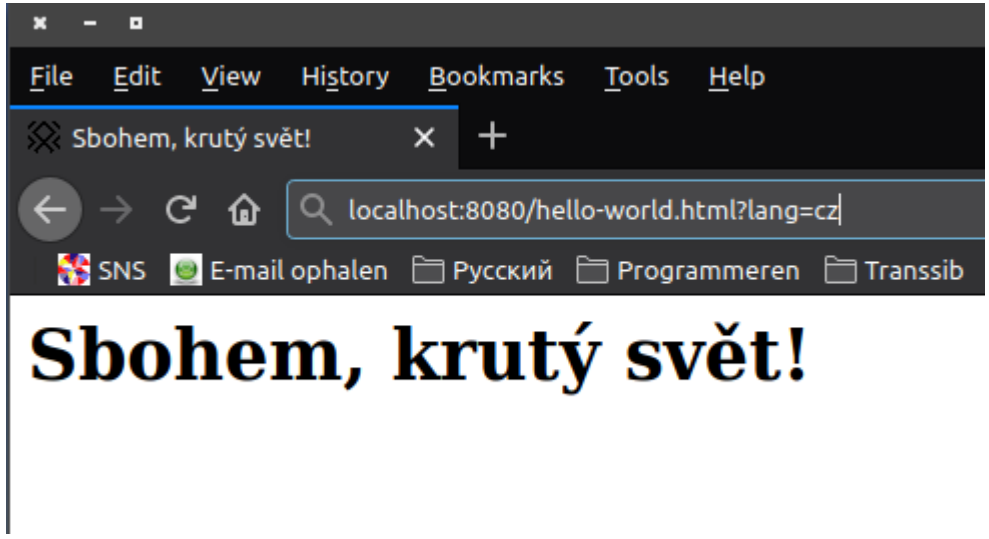




XSLWeb: the request dispatcher stylesheet

```
<xsl:template match="/req:request[req:path eq '/hello-world.html']">
    <xsl:variable name="lang"
    select="req:parameters/req:parameter[@name='lang']/req:value[1]"/>
    <pipeline:pipeline>
        <pipeline:transformer name="hello-world"
            xsl-path="{'hello-world-' || $lang || '.xsl'}"/>
    </pipeline:pipeline>
</xsl:template>
```





XSLWeb: the request dispatcher stylesheet

```
<xsl:variable name="reqparms" as="element(req:parameter)*"
    select="/req:*/req:parameters/req:parameter"/>

<xsl:template match="/req:request[req:path eq '/result-document']">
    <xsl:variable name="format" as="xs:string?"
        select="$reqparms[@name eq 'format']/req:value"/>

    <pipeline:transformer xsl-path="retrieve-xml.xsl"/>

    <xsl:choose>
        <xsl:when test="$format eq 'html'">
            <pipeline:transformer xsl-path="xml2html.xsl"/>
        </xsl:when>
        <xsl:when test="$format eq 'pdf'">
            <pipeline:transformer xsl-path="xml2fo.xsl"/>
            <pipeline:fop-serializer/>
        </xsl:when>
        <xsl:when test="$format eq 'epub'">
            <pipeline:transformer xsl-path="xml2epub.xsl"/>
            <pipeline:zip-serializer/>
        </xsl:when>
        <xsl:otherwise> .. error ..</xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

Apache FOP configuration in webapp.xml

XSLWeb: what makes a pipeline?

- One or more of the following **transformation** pipeline steps:
  - **transformer**: transforms the input of the pipeline step using an XSLT version 1.0, 2.0 or 3.0 stylesheet;
  - **query**: processes the input of the pipeline step using an XQuery version 1.0, 3.0 or 3.1 query;
  - **transformer-stx**: transform the input of the pipeline step using a STX (Streaming Transformations for XML) version 1.0 stylesheet.

XSLWeb: what makes a pipeline?

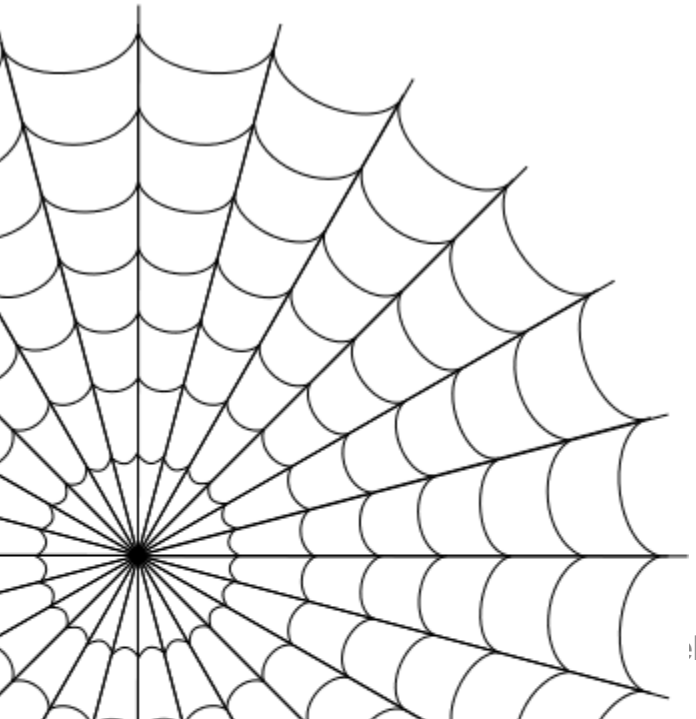
- Zero or more of the following **validation** pipeline steps:
  - **schema-validator**: validates the input of the step using an XML Schema version 1.0;
  - **schematron-validator**: validates the input of the step using an ISO Schematron schema.

XSLWeb: what makes a pipeline?

- Zero or one of the following **serialization** pipeline steps:
  - **json-serializer**: serializes XML output to a JSON representation;
  - **zip-serializer**: serializes an XML ZIP specification to an actual ZIP file;
  - **resource-serializer**: serializes a text or binary file to the response;
  - **fop-serializer**: serializes XSL-FO generated in a previous pipeline step to PDF using the Apache FOP XSL-FO processor.

XSLWeb: some goodies

- The **xslweb://** scheme can be used to call an internal XSLWeb pipeline from the XPath **doc()** or **document()** function.
- Request parameters **proxyHost** and **proxyPort** because sometimes you need to have them.
- Stylesheet parameters can be passed to stylesheets from the request-dispatcher stylesheet.



Why not XProc?

- There need not be just one solution for the same problem
- XSLWeb is very easy to learn
- XSLWeb lets you do things at the moment you want; no need to set up a second pipeline and merge its results
- XSLWeb has many facilities for frontend and webservice applications
- Creating XSLWeb was fun
- Using XSLWeb is fun

If you can use a bike,
you don't need a car

Why not XProc?

- There need not be just one solution for the same problem
- XSLWeb is very easy to learn
- XSLWeb lets you do things at the moment you want; no need to set up a second pipeline and merge its results
- XSLWeb has many facilities for frontend and webservice applications
- Creating XSLWeb was fun
- Using XSLWeb is fun

If you can use a bike,
you don't need a car

Why not XProc?

It's like XSLT and XQuery – they overlap

- Personally, I use Xproc:
  - In the case of large and complex content migrations, dealing with many documents, usually in batch
- I use XSLWeb:
  - In the case of relatively straighforward pipelines, especially in front-end or webservice applications
- Integrating XSLWeb and XProc is a serious option

Consider this – only one language to serve everything – XSLT on the server and – Saxon-JS – in the client! Life's a feast!

Questions?

Go ahead and try it.

Free and open source!

https://github.com/Armatiek/xslweb

Why XSLWeb

- Our Dutch government organisation has XML everywhere:
  - Government gazettes;
  - Legislation;
  - Search results (from Apache Solr, MarkLogic and other SRU interfaces);
  - RDF/XML;
  - Value lists;
  - SOAP/REST based webservices.

- Static generation of information is not an option for various reasons.

- Therefore we needed a web development framework with strong support for XSLT in which XSLT could be used to tie “everything XML” together.

- We wanted to have a framework that required **only XSLT knowledge** (no other scripting or programming language).

XSLWeb strength and weaknesses

- Strength of XSLWeb:
  - Only knowledge of XSLT is required (no other programming or scripting languages).
  - Open source, free to use
  - Suitable in environments where a lot of input data for the website or webservice is in XML format.
- Weaknesses of XSLWeb:
  - Pipelining not based on a standard (XProc).
  - Not a large community or developers (will that change today? ☺).

Why not Apache Cocoon?

- We used Apache Cocoon a lot, but:
  - the development of the product halted
  - we did not really like the rather static and declarative generator / transformer / serializer pattern.

If you want to write an application that communicates with a database or full text search engine, in Cocoon you need to:

- do all communication with that database in a generator step
- aggregate all data and then
- pass it on to one or more transformation steps.

But we needed a more dynamic approach, we wanted to communicate with the database from within the XSLT transformation steps itself. That's why the emphasis in XSLWeb is on the library of extension functions.

Why not … (fill in the blank)?

Other options could have been:

- eXist-db's URL Rewriting Controller
- RESTXQ (available in BaseX, eXist-db, FusionDB, and Marklogic)
- Marklogic's various controller.xq frameworks.

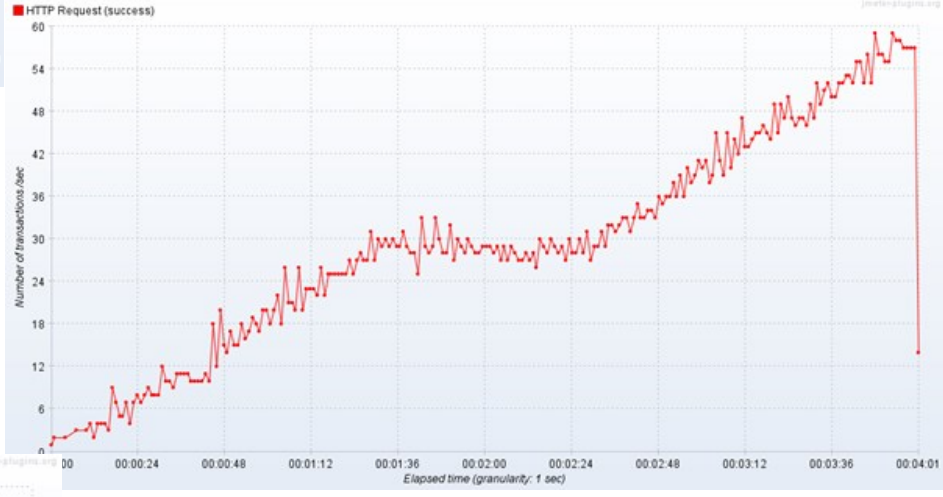
We use XSLWeb as a framework to build website and -service frontends for a variety of backends (Solr, MarkLogic, Microsoft FAST). We do not want it to be tied to one particular backend database like eXist/DB or BaseX (although we do use these databases and we very much like RestXQ, we do not use them in a production setting).

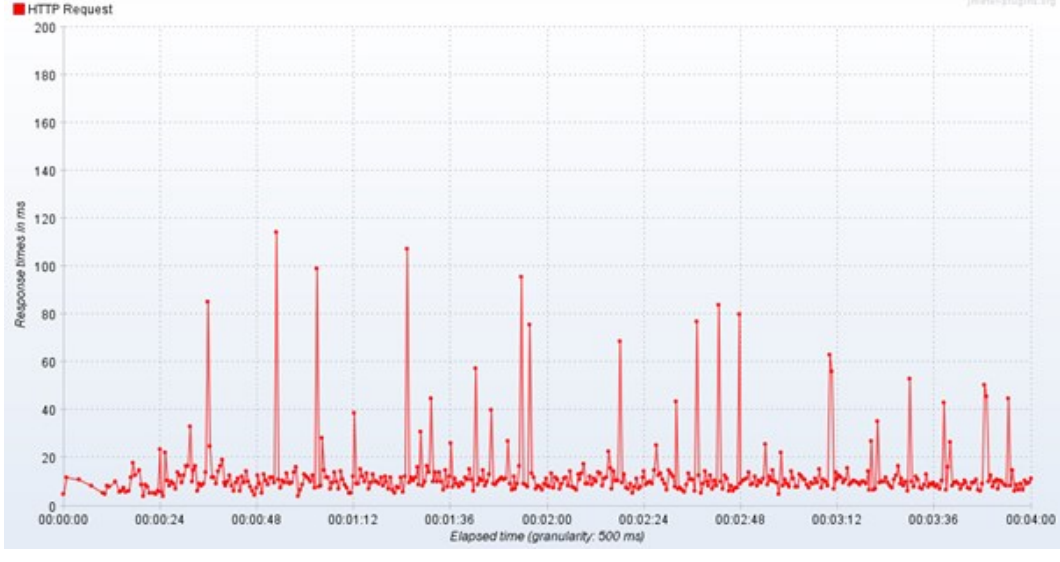
Performance requiremens

Performance requirements when we considered XSLWeb at the Dutch government:

- Minimum of 30 request/second
- Average response time not exceeding 150ms per request on a maximum of two HA Windows server VMs, Xeon CPU E5-2690, No SSD, 32Gb RAM
- Using a rather complex XSLT stylesheet that transforms Dutch government gazettes and legislation (in XML) to HTML.


Expected RPS
Number of requests/sec
Elapsed Time


HTTP Request (success)
Number of transactions/sec
Elapsed time (granularity: 1 sec)


HTTP Request
Response times in ms
Elapsed time (granularity: 500 ms)