EXPath Webapp

CXAN: a case-study for Servlex, an XML web framework

XML Prague, March 26th, 2011

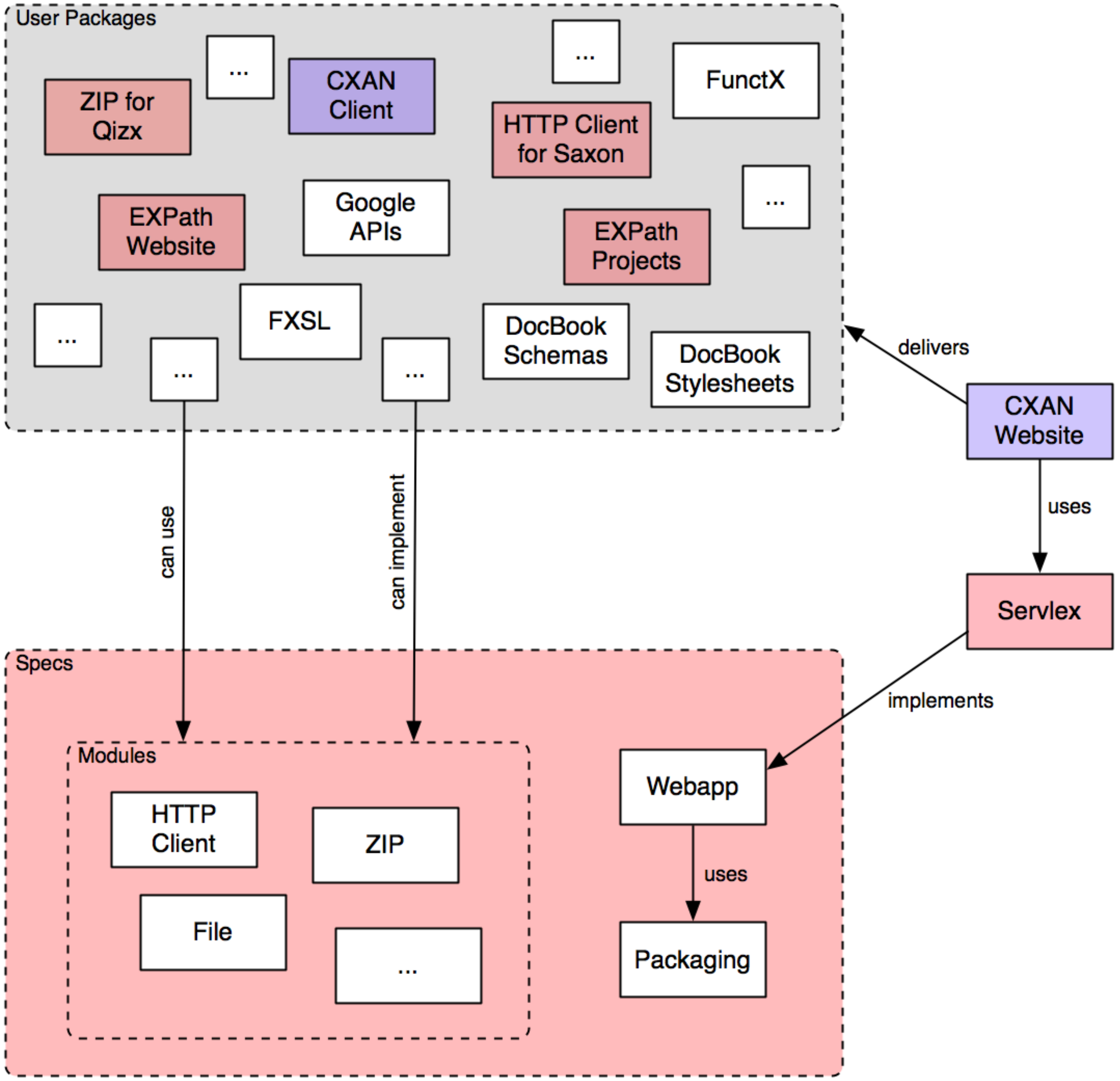
Florent Georges

H2O Consulting

EXPath Webapp

- **Introduction** ←
- Webapp, the module
- Data-driven
- Servlex
- The CXAN website
- From scratch
- Conclusion

The EXPath Universe



Introduction - Packaging

- A specification to package libraries
- Supported by several processors
  - eXist
  - Qizx
  - Saxon (as 3d party impl)
  - Calabash (as 3d party impl)
- A package is a standalone file...
- ... a ZIP file with components and a descriptor

Introduction - CXAN

- Same idea as CTAN for TeX/LaTeX, or CPAN for Perl, or APT for Debian
- A central, comprehensive, organized collection of packages:
  - Libraries (XSLT, XQuery, XProc, schemas, …)
  - Applications (command line, webapps, …)
- Maintained on a central website
- Accessible through a browser
- Accessible through the CXAN client

Introduction - Webapp

- Define a web container to deploy webapps
- A webapp is a set of components (in XSLT, XQuery or XProc)
- It has a descriptor to map incoming URIs to specific components
- It is packaged using the Packaging System
- A webapp is able to do whatever is possible to do with HTTP on server-side
- That is, it is able to build *your* website

Introduction - Samples

- An existing implementation of EXPath Webapp is Servlex
- Some existing websites deployed on Servlex:
  - http://expath.org/
  - http://h2oconsulting.be/
  - and... http://cxan.org/
- Of interest:
  - http://h2oconsulting.be/xqts/
  - http://h2oconsulting.be/tools/dump

EXPath Webapp

- Introduction
- **Webapp, the module** ←
- Data-driven
- Servlex
- The CXAN website
- From scratch
- Conclusion

Webapp - Goal

- The final, user-level goal is to be able to respond to any HTTP request on the server (so implementing websites, web services, etc.)
- Webapp is aimed at defining a processor-agnostic web container for XML technologies
- The web container allows one to deploy web applications written directly in XSLT, XQuery and/or XProc
- A webapp is interoperable as long as it respects the rules in the Webapp spec

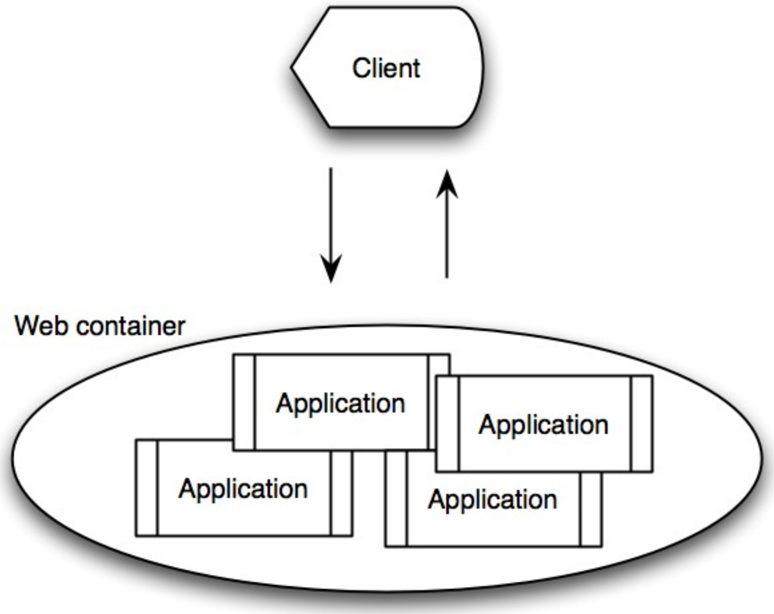
H2O Consulting

Webapp - Components

- Components are either:
  - XProc step
  - XProc pipeline
  - XQuery function
  - XQuery main module
  - XSLT function
  - XSLT named template
  - XSLT stylesheet

Webapp - At 10,000 feet

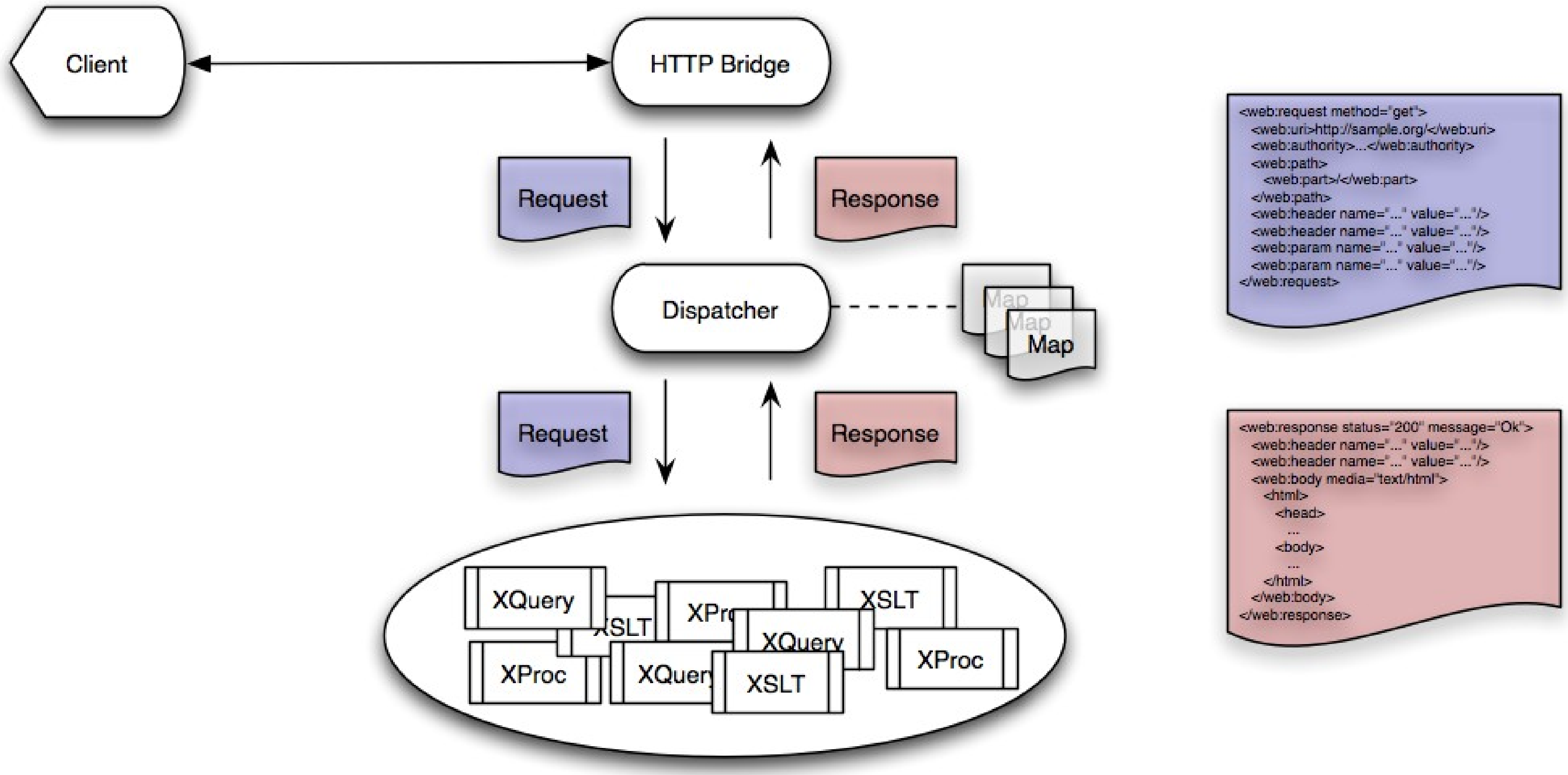
- The Webapp Module defines a *web container* responding to HTTP requests
- The processing is defined by user applications



Webapp - Overall processing

- Under the hood, the URI of each HTTP request is inspected
- It is used to choose an application to serve it
- It is then used to pick a component within the application, using its own map
- The component is evaluated with an XML representation of the HTTP request
- The component result is an XML rep of the HTTP response to send back

Webapp - 1:1 scale

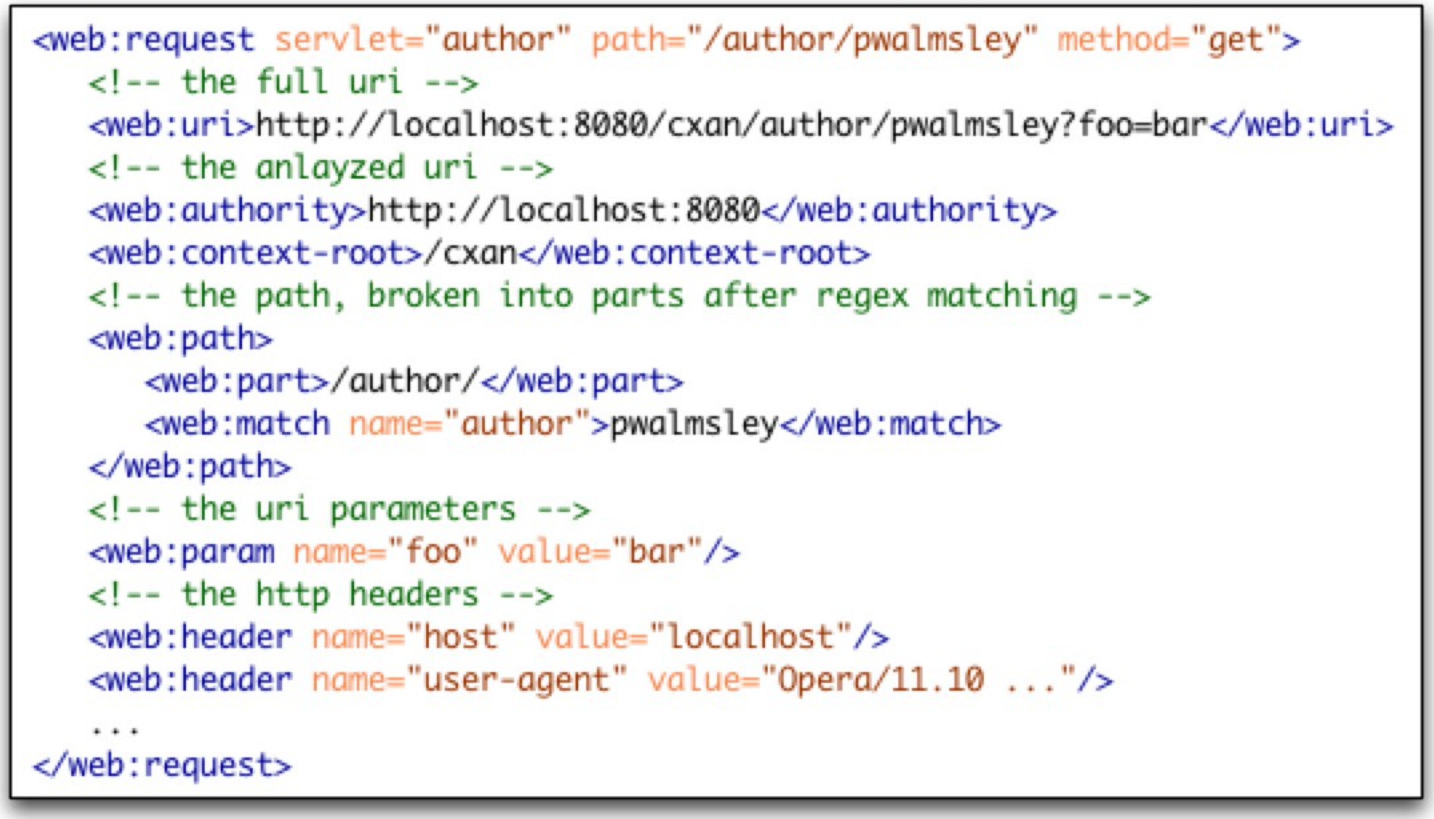


```
<web:request method="get">
  <web:uri>http://sample.org/</web:uri>
  <web:authority>...</web:authority>
  <web:path>
    <web:part>/</web:part>
  </web:path>
  <web:header name="..." value="..."/>
  <web:header name="..." value="..."/>
  <web:param name="..." value="..."/>
  <web:param name="..." value="..."/>
</web:request>
```

```
<web:response status="200" message="Ok">
  <web:header name="..." value="..."/>
  <web:header name="..." value="..."/>
  <web:body media="text/html">
    <html>
      <head>
        ...
      <body>
        ...
    </html>
  </web:body>
</web:response>
```

Webapp – The request

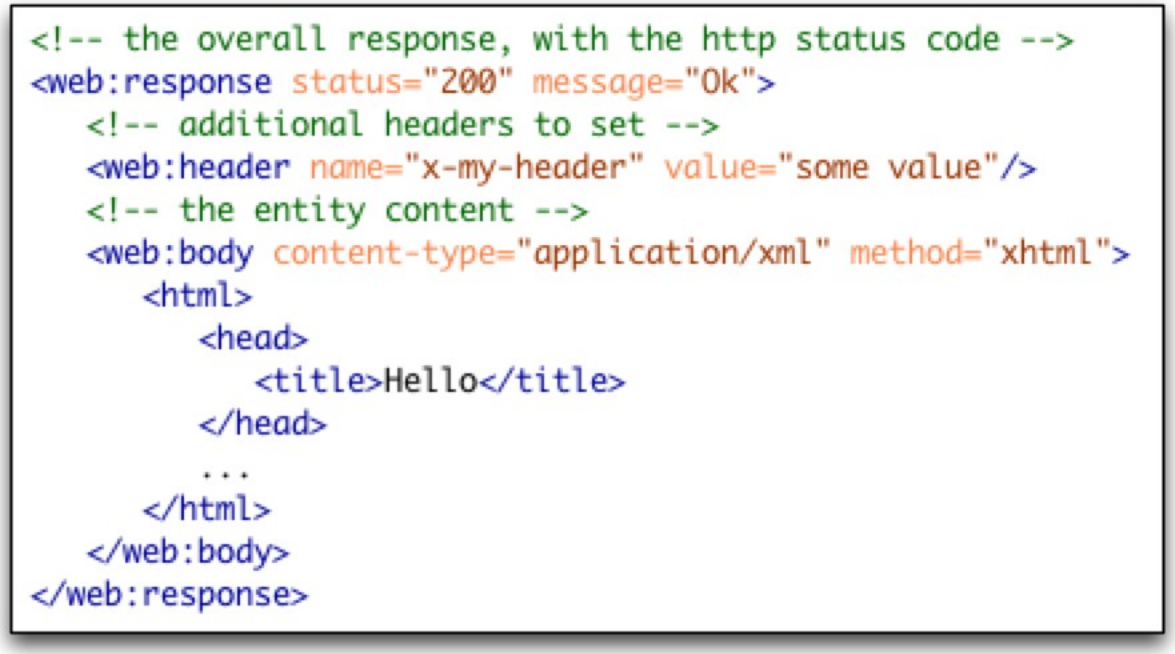
- Represent the HTTP request in XML
- Easy access to specific pieces using XPath



```xml
<web:request servlet="author" path="/author/pwalmsley" method="get">
   <!-- the full uri -->
   <web:uri>http://localhost:8080/cxan/author/pwalmsley?foo=bar</web:uri>
   <!-- the anlayzed uri -->
   <web:authority>http://localhost:8080</web:authority>
   <web:context-root>/cxan</web:context-root>
   <!-- the path, broken into parts after regex matching -->
   <web:path>
      <web:part>/author/</web:part>
      <web:match name="author">pwalmsley</web:match>
   </web:path>
   <!-- the uri parameters -->
   <web:param name="foo" value="bar"/>
   <!-- the http headers -->
   <web:header name="host" value="localhost"/>
   <web:header name="user-agent" value="Opera/11.10 ..."/>
   ...
</web:request>
```

Webapp – The response

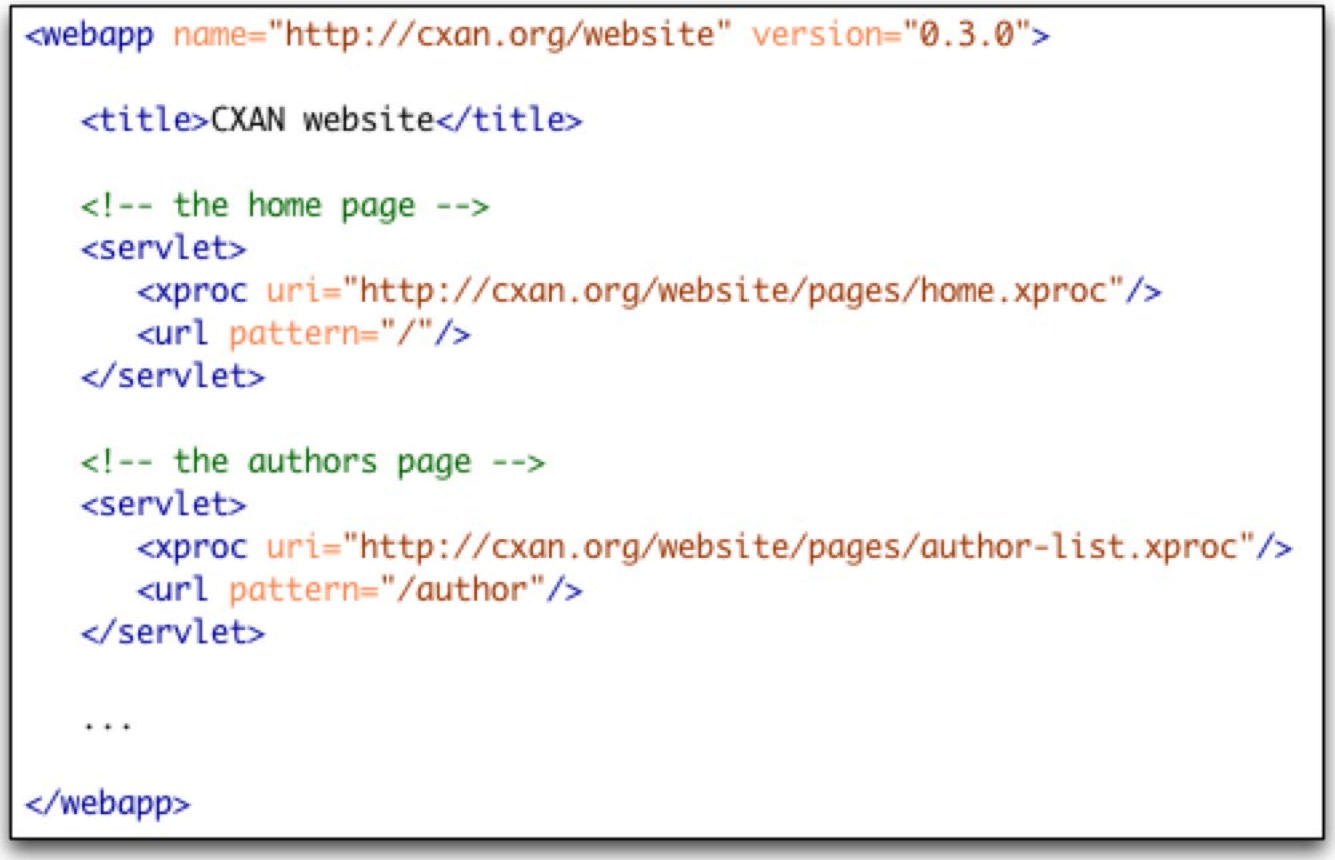
- Represent the HTTP response in XML
- Returned by the component called with the request element



```
<!-- the overall response, with the http status code -->
<web:response status="200" message="Ok">
   <!-- additional headers to set -->
   <web:header name="x-my-header" value="some value"/>
   <!-- the entity content -->
   <web:body content-type="application/xml" method="xhtml">
      <html>
         <head>
            <title>Hello</title>
         </head>
         ...
      </html>
   </web:body>
</web:response>
```

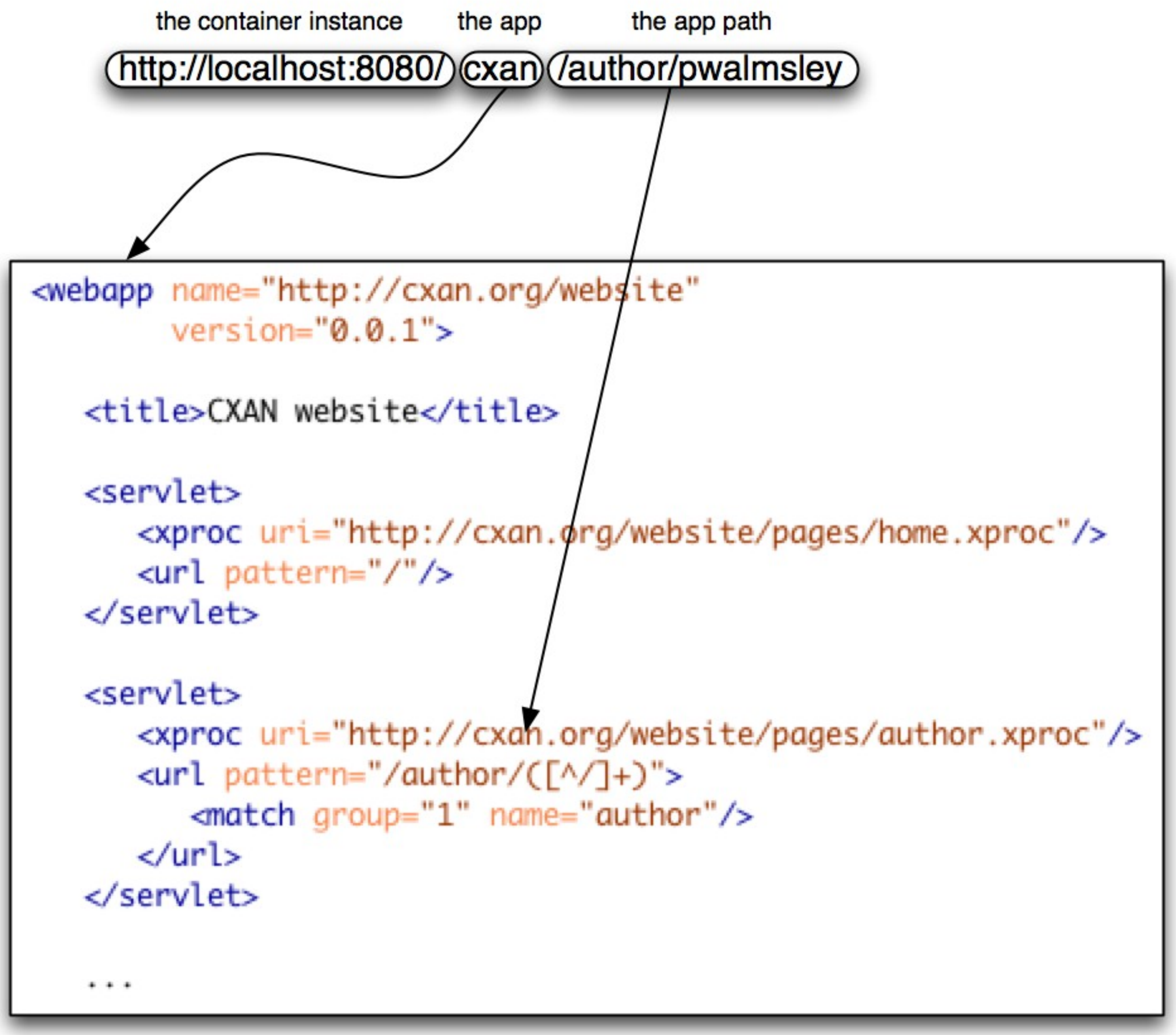
Webapp – The dispatcher

- Based on the web descriptor
- It maps URI to components, using regexs



```
<webapp name="http://cxan.org/website" version="0.3.0">

   <title>CXAN website</title>

   <!-- the home page -->
   <servlet>
      <xproc uri="http://cxan.org/website/pages/home.xproc"/>
      <url pattern="/"/>
   </servlet>

   <!-- the authors page -->
   <servlet>
      <xproc uri="http://cxan.org/website/pages/author-list.xproc"/>
      <url pattern="/author"/>
   </servlet>

   ...

</webapp>
```

Webapp – The dispatcher



```
<webapp name="http://cxan.org/website"
        version="0.0.1">

   <title>CXAN website</title>

   <servlet>
      <xproc uri="http://cxan.org/website/pages/home.xproc"/>
      <url pattern="/"/>
   </servlet>

   <servlet>
      <xproc uri="http://cxan.org/website/pages/author.xproc"/>
      <url pattern="/author/([^/]+)">
         <match group="1" name="author"/>
      </url>
   </servlet>

   ...
```

Webapp - Filters & Co.

- The dispatcher can perform more complex processing
- According to the map, it can setup filters, error handlers, chain them, and order them
- Error handlers help setting a consistent error reporting mechanism
- Filters can apply a consistent layout, inject data, handle authentication, or build a dedicated data model for the application

Webapp - Packaging

- Webapp defines how to package a webapp
- It builds on the EXPath Packaging System
- Basically, a webapp file is a standard package with a web descriptor, expath-web.xml
- Standard packaging tools can be used for web applications
- A webapp itself can be published on CXAN, thus retrieved and installed automatically in a web container's repository

EXPath Webapp

- Introduction
- Webapp, the module
- **Data-driven** ←
- Servlex
- The CXAN website
- From scratch
- Conclusion

Data - What's different?

- The framework is data-oriented, in particular the web:request and web:response elements
- What is different? Nothing, one could argue
- But in the developer's eyes
  - More clear, easier to learn
  - Easier to log, and to investigate bug reports
- Easy to create:
  - A custom data layer dedicated to the app
  - Filters transforming inputs & outputs

Data - Testability

- The web:request and web:response elements are the interface between the application components and the web container
- Because they are data, it is easy to create them by hand or generate them, and to save them on disk and in a revision control system
- Even at the highest level, no need of plenty of dedicated functional test tools to simulate the behavior of the web container, a web:request element describes a complete HTTP request

H2O Consulting

Data - Testability, say it again

A webapp is unit-testable all along the way, up to the very top of the stack

EXPath Webapp

Servlex - Intro

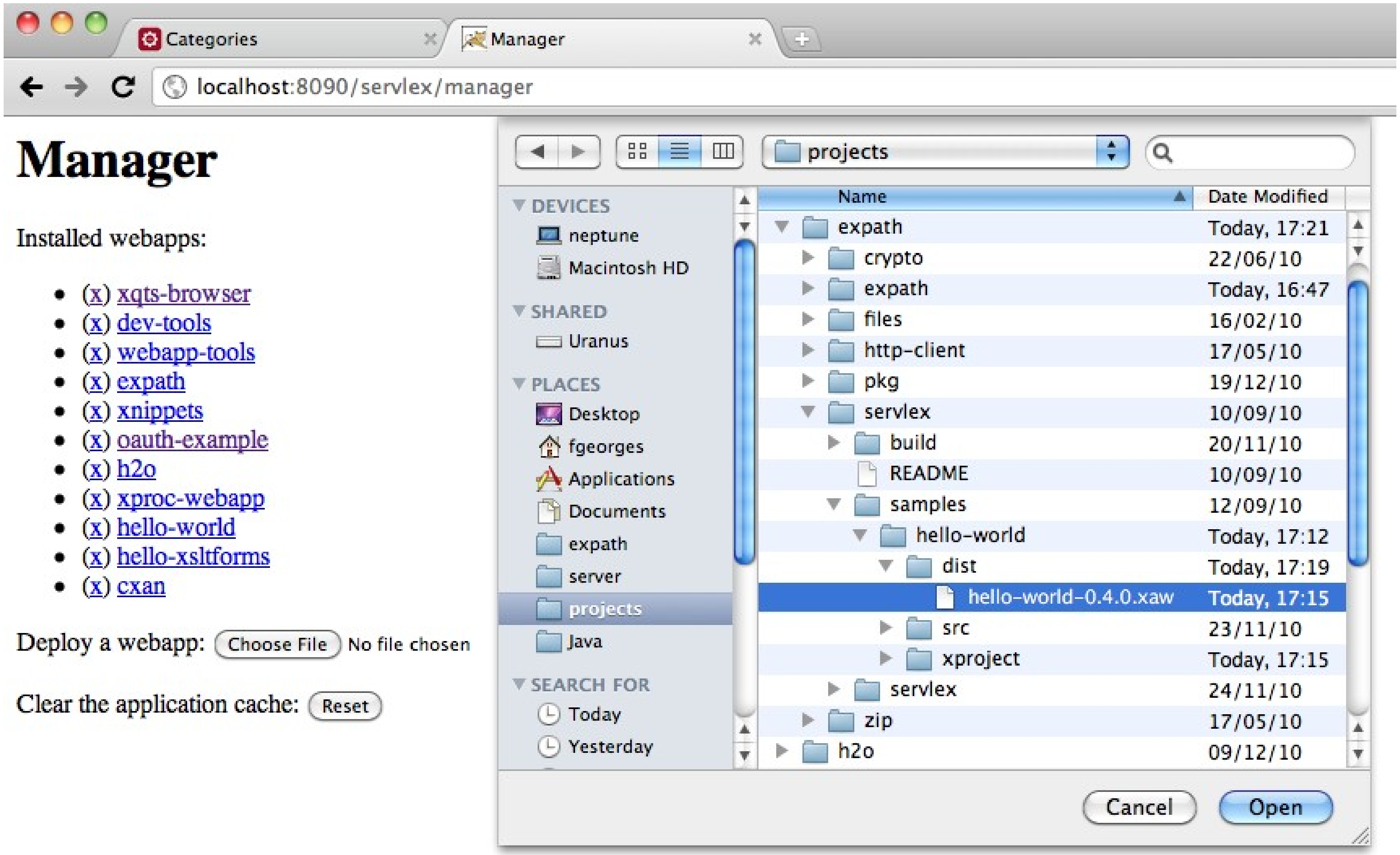
- Servlex in one implementation of the web container defined in the Webapp Module
- It is open-source
- It is available at http://servlex.googlecode.com/
- It is written in Java
- On the network hand, it uses Java Servlet technology, for the link to HTTP
- On the XML hand, it uses Saxon and Calabash, as its XSLT, XQuery and XProc processors

H2O Consulting

Servlex - Install

- Servlex is a standard WAR file
- It can be deployed in Tomcat, Jetty, Glassfish, but also in Google Appengine and Amazon Cloud EC2 (you know, that cloud thing)
- The only config is to point to a standard on-disk package repository (by setting a Java system property)
- It contains a admin interface to manage and install web applications and packages

Servlex - Manager



Categories | Manager

localhost:8090/servlex/manager

Manager

Installed webapps:

- (x) xqts-browser
- (x) dev-tools
- (x) webapp-tools
- (x) expath
- (x) xnippets
- (x) oauth-example
- (x) h2o
- (x) xproc-webapp
- (x) hello-world
- (x) hello-xsltforms
- (x) cxan

Deploy a webapp: Choose File No file chosen

Clear the application cache: Reset

Name	Date Modified
expath	Today, 17:21
crypto	22/06/10
expath	Today, 16:47
files	16/02/10
http-client	17/05/10
pkg	19/12/10
servlex	10/09/10
build	20/11/10
README	10/09/10
samples	12/09/10
hello-world	Today, 17:12
dist	Today, 17:19
hello-world-0.4.0.xaw	Today, 17:15
src	23/11/10
xproject	Today, 17:15
servlex	24/11/10
zip	17/05/10
h2o	09/12/10

Servlex - Google Appengine

- Google Appengine does not allow disk access
- The repository is all in the classpath
- The set of webapps is therefore fixed in the Servlex instance deployed on Appengine
- http://fgeorges.appspot.com/expath/ as an example
- Could be deployed in other cloud systems

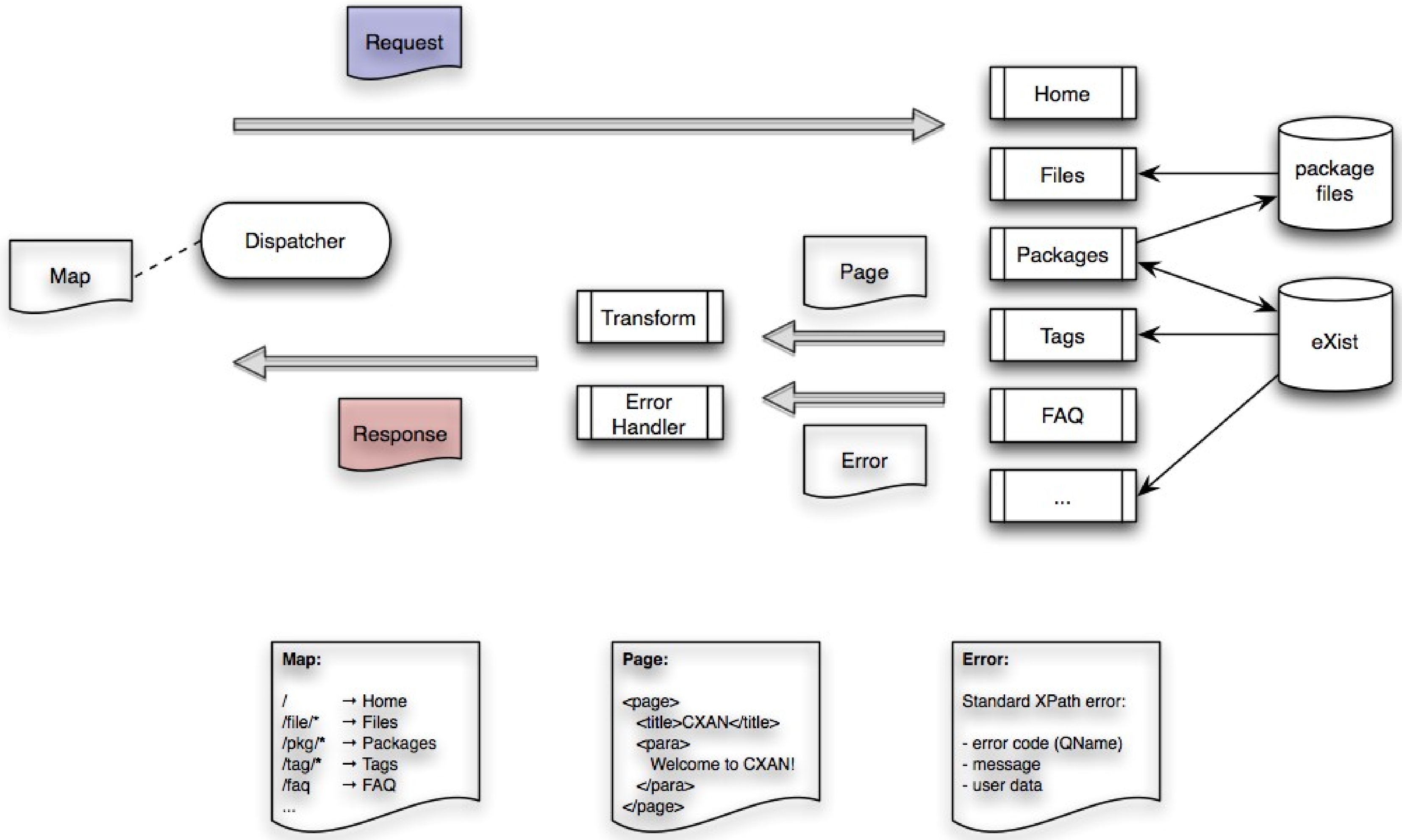
EXPath Webapp

- Introduction
- Webapp, the module
- Data-driven
- Servlex
- **The CXAN website** ←
- From scratch
- Conclusion

CXAN - The website

- The central part of CXAN is its website
- It can be browsed as a website or used via its well-defined REST-like API
- There is a command-line client, but there can be several of them, e.g. in an XML database
- It stores packages (XAR files + distributions)
- It organizes them by name, and by using tags, categories, and authors, as well as a CXAN ID
- A package stores additional infos in cxan.xml

CXAN - Architecture



CXAN - Architecture

- One component for each page (in XProc)
- There is a global error handler, catching XPath errors, returning a user-friendly error page
- The components return a page document
- There is a global output filter, transforming page documents into full HTML pages, with consistent layout
- Based on the HTTP Accept header, they can return XML instead (for REST-style calls)

EXPath Webapp

- Introduction
- Webapp, the module
- Data-driven
- Servlex
- The CXAN website
- **From scratch** ←
- Conclusion

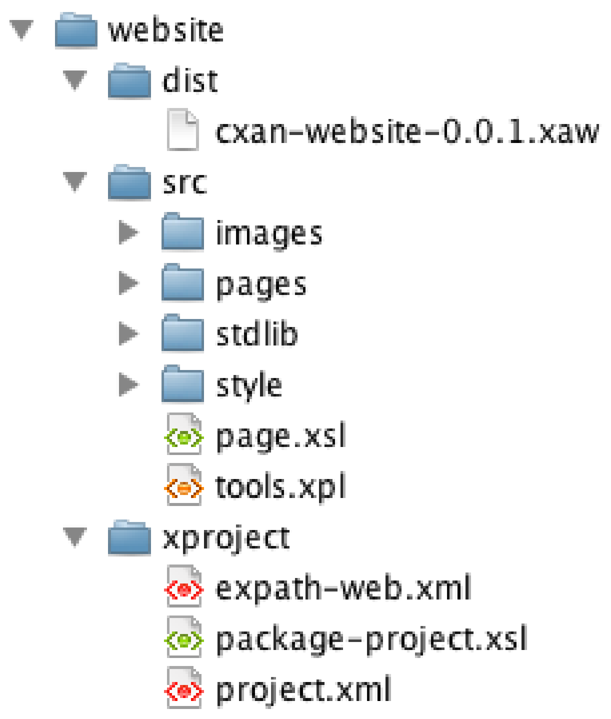
From scratch - Old school

- A webapp is a standard package
- So it is a ZIP file, with a particular structure
- So it must contain a package descriptor
- The package descriptor identifies components (by assigning them public import URIs)
- It contains a web descriptor (expath-web.xml), linking request URIs patterns to components

From scratch - New age

- Maintaining the package descriptor separately from the components is painful
- So is creating the ZIP file
- A standard tool, based on some project infos, annotations in the components and a project structure, builds the package descriptor and the ZIP file
- We have to write the web descriptor, be we could improve the tool to use annotations

From scratch - Project structure

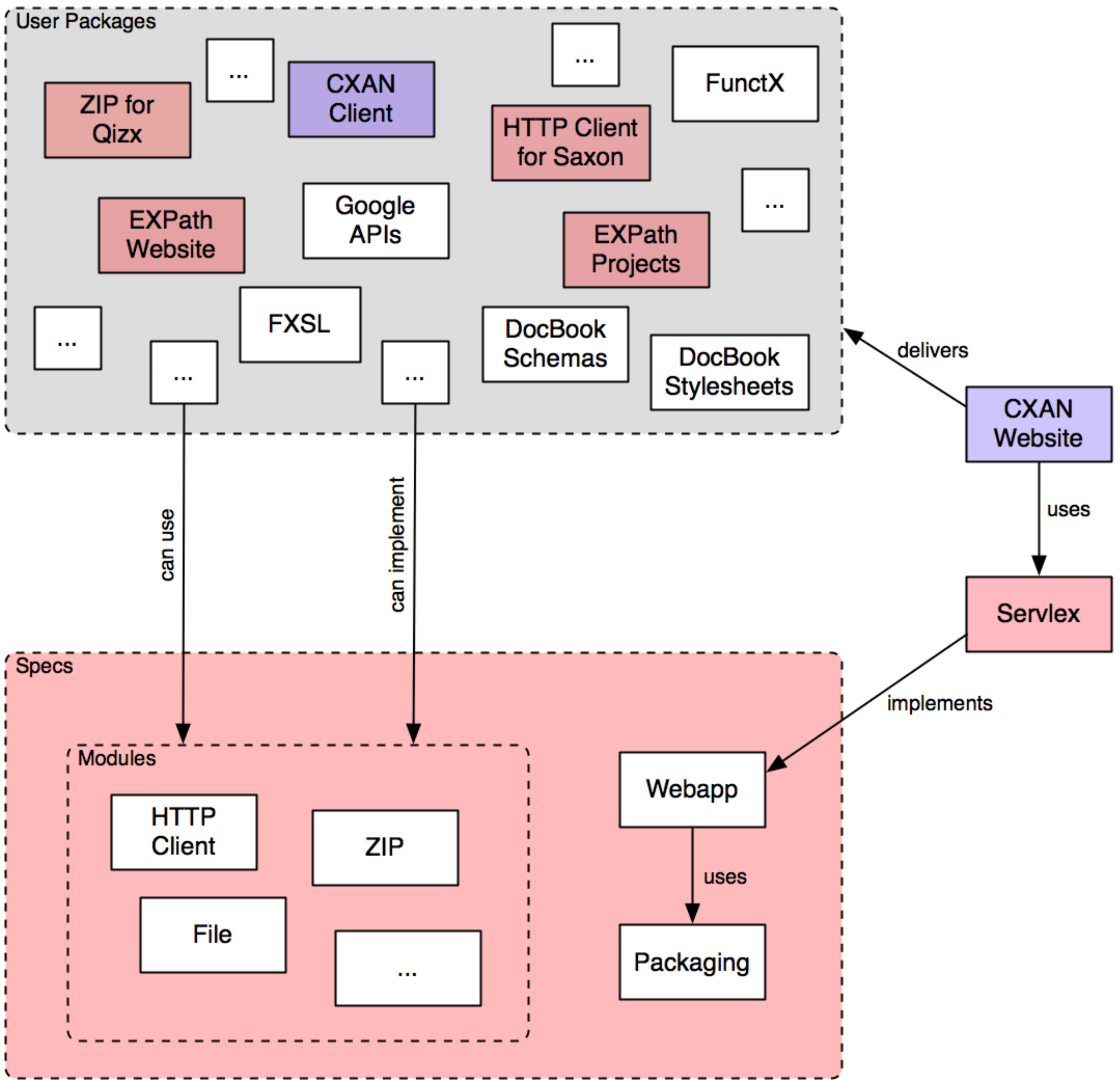


```
▼ website
  ▼ dist
      cxan-website-0.0.1.xaw
  ▼ src
    ▶ images
    ▶ pages
    ▶ stdlib
    ▶ style
      page.xsl
      tools.xpl
  ▼ xproject
      expath-web.xml
      package-project.xsl
      project.xml
```

EXPath Webapp

- Introduction
- Webapp, the module
- Data-driven
- Servlex
- The CXAN website
- From scratch
- **Conclusion** ←

EXPath Universe - Once again



- Join the mailing list and install Servlex:

  http://expath.org/

  http://servlex.googlecode.com/