Seamless Content Management with OpenOffice and Cocoon

Christian Egli

1.0

Table of Contents

Abstract

XML is establishing itself as the universal content format. Not only is it widely used in web publishing but it is also becoming the lingua franca for office tools such as OpenOffice.org and future versions of Microsoft Office.

For the purpose of publishing content from a single XML source to multiple output formats, Apache Cocoon is an excellent tool. It provides powerful and flexible mechanisms to publish XML content in almost any format.

Although there are many tools to aid in the creation of XML content this task is still hard. What the user wants is not an XML editor where she inserts elements. No, what she really wants is a WYSIWYG tool that is integrated in the office environment (and still provides all the benefits of validation, separation of content and presentation, etc.). Yes, there are browser-based solutions that work quite well, but none of them match the maturity and wide acceptance of an office suite.

That's where OpenOffice.org comes to the rescue. OpenOffice.org is a widely used open source cross-platform office suite which uses XML as the native file format for all its documents. The format is documented and open.

Combining OpenOffice.org and Apache Cocoon provides us with a powerful tool chain for creating and publishing content. If we add WebDAV to the mix, we also get distributed content authoring.

This paper will present the tools outlined and give a brief overview of the history and fundamental concepts behind them. A possible implementation of the combination will be presented and directions for further exploration outlined.

Problem

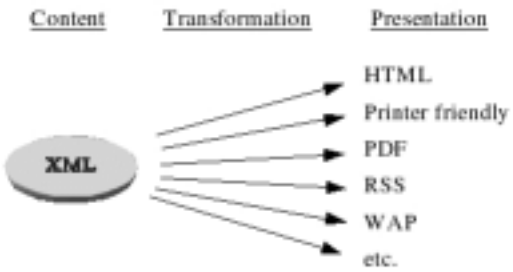
Historically, content has been produced in conjunction with presentation, i.e. content is tightly coupled with presentation. Probably the best known example of this practice is HTML where tables are typically used for layout and the content is usually littered with font and other presentational attributes. Other examples include Microsoft Word.

While this might be a easy and convenient solution, it does not scale to today's demands where the same content is to be published in numerous formats, e.g. plain HTML page, printer friendly HTML page, possibly PDF, maybe RSS.

Separate content from presentation

The obvious solution to this problem is to separate content from presentation. When we do this we end up with a document containing the pure content and any number of separate transformation descriptions for each particular presentation. A transformation description for a book would for example outline where to put page numbers and page breaks, whereas a transformation description for a web page might specify where to put banner ads or navigational elements.

To produce a particular presentation, for example HTML, we simply take the content and apply the transformations as they are described in the transformation descriptions for the HTML presentation. This allows us to produce any presentational format from one single content source as shown in Figure 1.

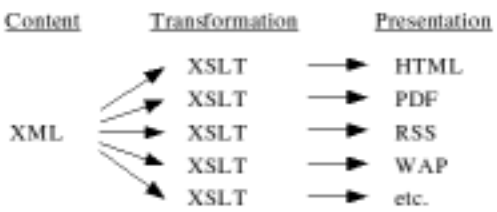


Enabling technology

XML, XSLT and other World Wide Web Consortium (W3C) standards have established themselves as key enabling technologies to achieve the separation of content and presentation and provide an easy and standard transformation mechanism between content and presentation.

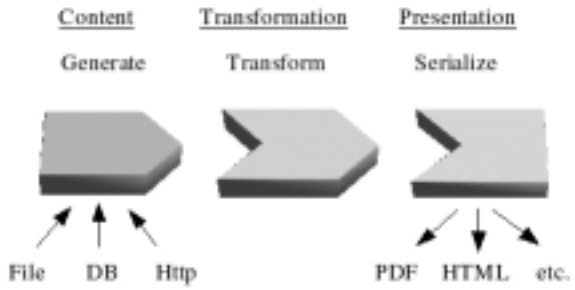
XML (eXtensible Markup Language) is a standard based on SGML and is increasingly being used as the universal, presentation-free content format. It lends itself perfectly for media-independent publishing and data interchange. It is widely used in web publishing and is also becoming the lingua franca for office tools such as OpenOffice.org and future versions of Microsoft Office.

XSLT (Extensible Stylesheet Language Transformations) is a language for transforming XML documents. It allows transformation of one XML format into another (e.g. XHTML, RSS) or into any other text-based format as illustrated in Figure 2.

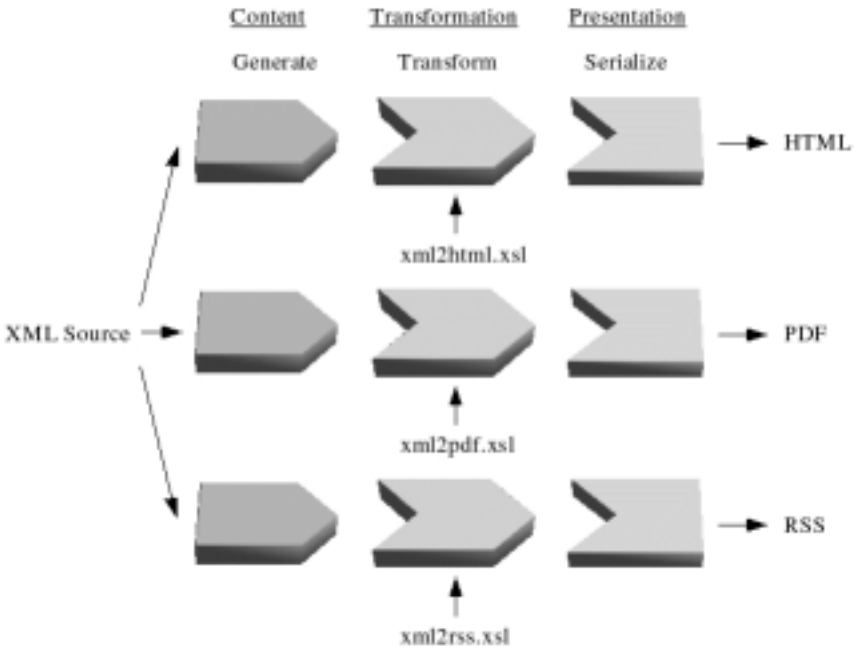


Apache Cocoon

Apache Cocoon is a highly flexible open-source web publishing framework that is based on XML and XSLT. One of the fundamental concepts is the XML pipeline, where XML data is passed between the components. An example of a very basic pipeline is shown in Figure 3. The first component, a so-called generator, produces an XML stream from a file, a database or other sources. The XML is then piped into a transformer which manipulates it, usually by applying an XSL transformation to it. Finally, the XML data is passed to a serializer which converts the XML stream into the format required by the requesting client, e.g. HTML, PDF, JPEG, etc. Figure 3 illustrates not only the concept of the Cocoon pipeline, but also shows how Cocoon supports separation of content and presentation in a very straight-forward way.



The concept of transformation from single content to multiple presentation as depicted in Figure 2 maps directly to the Cocoon concept of pipelines. Each arrow in Figure 2 symbolizes a transformation of the content to a particular presentation. In Cocoon this translates to multiple pipelines where the corresponding transformation for each presentation is applied as can be seen in Figure 4.



Example

The pipelines in Cocoon are defined in the so-called sitemap which itself is an XML file. The following snippet from a sitemap shows the pipelines as they correspond to Figure 4:

```
<map:pipeline>
  <map:match pattern="simple.html">
    <map:generate src="simple.xml"/>
    <map:transform src="xml2html.xsl"/>
    <map:serialize type="html"/>
  </map:match>

  <map:match pattern="simple.pdf">
    <map:generate src="simple.xml"/>
    <map:transform src="xml2pdf.xsl"/>
    <map:serialize type="fo2pdf"/>
  </map:match>

  <map:match pattern="simple.rss">
    <map:generate src="simple.xml"/>
    <map:transform src="xml2rss.xsl"/>
    <map:serialize type="xml"/>
  </map:match>

</map:pipeline>
```

Cocoon uses the concept of matchers. For each request Cocoon determines which pipeline ought to be used to handle it. In the example above very simple pattern matchers are used. The request for "simple.html" is obviously handled by the first pipeline which delivers HTML since the request exactly matches the pattern of the matcher. Requests for "simple.pdf" and "simple.rss" are handled accordingly.

Content Creation

We have seen how Cocoon can help with publishing XML content. However, when it comes to creating the content, we are faced with editing XML. Although it is human-readable, it is not something that people beyond the command-line and vi/Emacs crowd are willing to look at. The majority of users are used to producing content with WYSIWYG "Office Applications". They also expect their content production tools to be integrated seamlessly in the desktop so that features like drag-and-drop just work.

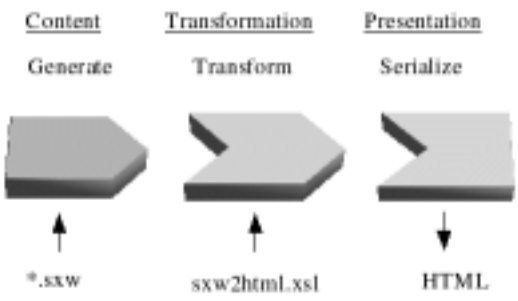
As it happens, OpenOffice.org, a free multi-platform office productivity suite, uses XML as its native file format.

Would it be possible to combine OpenOffice.org for content creation and Cocoon for content publication?

OpenOffice.org

There is hardly a need to introduce OpenOffice.org. It includes the key desktop applications, such as a word processor, spreadsheet, presentation manager, and drawing program, with a user interface and feature set similar to other office suites'.

Since its native file format is XML, open and very well documented, OpenOffice.org lends itself to being a major player in the XML publishing tool chain as shown in the conceptional overview in Figure 5.



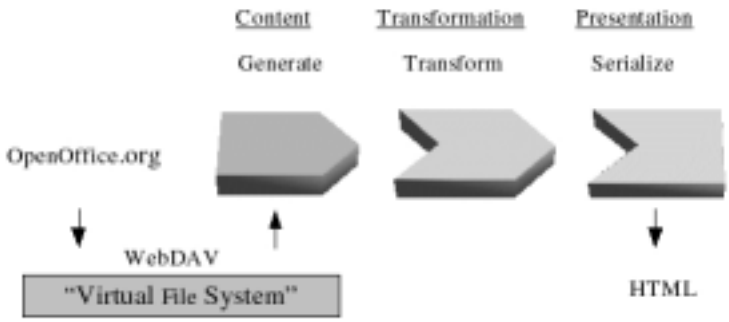
As in to Figure 3, Figure 5 also shows a pipeline with an XML file, in this case an OpenOffice.org file (*.swx), at the beginning. To produce HTML, an XSL transformation is applied to this XML and finally it is serialized as HTML.

The actual implementation of the publishing process is not as straight-forward as the conceptional overview suggests. The OpenOffice.org file is actually a zip file containing multiple XML files, and possibly other files such as images in JPEG or GIF format. In order to feed the Cocoon pipeline one of the XML files (`content.xml`) is extracted from the zip file and forwarded for XSL transformation. For the technical details of the actual implementation please refer to the descriptions in the appendix.

WebDAV

XML publishing, especially when it comes to web publishing, is typically regarded as a task that is handled on the server. On the other hand, content creation is typically done on a desktop machine. To achieve the touted seamless integration we need to find a way to make this distinction transparent.

The ideal solution for this is WebDAV (Web-based Distributed Authoring and Versioning). WebDAV provides a set of extensions to the HTTP protocol which allows users to edit and manage files on remote web servers collaboratively.



This solution allows content creators to edit their content with OpenOffice.org and seemingly store it on their desktop machine. In reality, of course, it stored on the server from where it can be served by Cocoon in multiple output formats.

There is more to Content Management

While we have shown that the combination of OpenOffice.org and Cocoon can provide a solution for the needs of content creation and delivery, the combination lacks support for another crucial part of content management as defined in the feature list of OSCOM, the organization for open source content management: the management proper of the content. Content management includes aspects such as workflow, access control, versioning (e.g. scheduling, expiration), personalization and localization.

Some of these features, like access control can be provided by WebDAV. Others, such as versioning, will be provided by WebDAV in the future. Personalization and localization can be partially handled by Cocoon, but these features don’t cover the whole range of requirements as specified by OSCOM.

The real answer is of course to use a proper Content Management System. The ideal would be a Cocoon-based solution which would allow us to leverage its XML publishing strengths. A number of up and coming contenders hope to fill the role of a Cocoon-based Content Management System. One of the most promising candidates, Apache Lenya, has just recently been accepted as an official Apache subproject .

Conclusion

We have seen that OpenOffice.org and Cocoon can complement each other in providing a powerful solution for content creation and delivery in multiple presentations.

WebDAV can enhance this by providing transparent access to content on remote servers.

More work is needed in the area of content management proper when it comes to versioning, access control and workflow. Some of these needs can be handled by WebDAV and additions to Cocoon, but to provide the full range of features for what is generally called "content management", this is not enough.

The current choices available for the content publisher are:

- to use a "real" Content Management System (CMS) which has features like workflow and localization. Currently, however, there is no full-fledged CMS based on Cocoon so some of the major benefits like content-presentation separation and multi-presentation delivery might not be provided by the "real" CMS.
- to wait for WebDAV to deliver on its promises to standardize and implement versioning, configuration management and access control.
- to use one of the up and coming Cocoon-based CMS like Apache Lenya.

References

- Cocoon: http://cocoon.apache.org
- OSCOM: http://www.oscom.org
- OpenOffice.org: http://www.openoffice.org
- W3C: http://www.w3.org
- WebDAV:
  - http://www.ics.uci.edu/~ejw/authoring
  - http://www.webdav.org
  - http://webservices.xml.com/pub/a/ws/2002/03/26/webdav.html
- XML: http://www.xml.com/pub/a/98/10/guide1.html
- XSLT:
  - http://www.w3.org/Style/XSL
  - http://www.xml.com/pub/a/2000/08/holman

A. Appendix

Appendix

Prerequisites

In order to be able to publish OpenOffice.org documents with Cocoon the following prerequisites have to be met:

• OpenOffice.org DTD

The DTDs for OpenOffice documents have to be installed on the system. They can be found in the local OpenOffice.org installation.

• XML Catalog

In order for Cocoon to find the DTDs an XML catalog has to be setup. The `CatalogManager.properties` file has to be edited for Cocoon to know about this new XML catalog.

• OpenOffice2HTML XSLT

In order to render the OpenOffice XML as HTML, the XSL transformation file has to be installed where Cocoon can find it. A very good XSLT which is fairly complete can be fetched from zope.org (http://www.zope.org/Members/philikon/ZooDocument).

Cocoon Pipelines

The following describes the pipelines that have to be set up in order to be able to serve OpenOffice.org documents with Cocoon.

• Read the zip/jar file

A very simple pipeline is needed which basically just reads the OpenOffice document and returns it as is.

```
<map:match pattern="**.sxw">
  <map:read src="content/{1}.sxw"/>
</map:match>
```

• Unpack zip file and transform the OpenOffice.org XML to HTML

OpenOffice.org documents are zip files containing XML files for content and style plus optionally additional files such as JPEG or PNG. The actual content is contained in a file named `content.xml`. In order to present the OpenOffice.org document Cocoon has to unpack the zip file, extract the `content.xml` file and apply the XSL transformation to render it as HTML

Zip is the same file format as jar. Java supports jar unpacking natively with the jar protocol. The pipeline which reads the jar file, extracts the content and applies the transformation, looks as follows:

```
<map:match pattern="**.oo">
  <map:generate src="jar:http://localhost:8080/{1}.sxw!/content.xml"/>
  <map:transform src="xslt/ooo2html.xsl"/>
  <map:serialize/>
</map:match>
```