

---

# **XML**

# **A New Web Site Architecture**

---

**Jim Costello  
Derek Werthmuller  
Darshana Apte**

**Center for Technology in Government  
University at Albany, SUNY  
1535 Western Avenue  
Albany, NY 12203  
Phone: (518) 442-3892  
Fax: (518) 442-3886  
E-mail: [info@ctg.albany.edu](mailto:info@ctg.albany.edu)  
<http://www.ctg.albany.edu>**

**September 2002**

---

# Table of Contents

XML: A New Web Site Architecture .....	1
A Better Way? .....	1
Defining the Problem .....	1
Partial Solutions .....	2
Addressing the Root Problems .....	2
Figure 1. Sample XML file (all code simplified for example) .....	4
Figure 2. Sample XSL File (all code simplified for example) .....	6
Figure 3. Formatted Page Produced from Sample XML File using XSL .....	7
Workflow Impact .....	7
Figure 4. Workflow (Simplified) in HTML-based Web Site Management .....	8
Figure 5. Workflow (Simplified) in XML-based Web Site Management .....	9
Working with XML .....	9
Converting Our Site .....	10
Figure 6. Transforming Documents from PDF to RTF to XML .....	11
The Advantage of an XML Document .....	11
Transforming Content with XSL .....	13
Figure 7. Connecting XML and XSL Files .....	13
Figure 8. Transforming XML Documents into XHTML and PDF .....	14
Maintaining Style Consistency .....	14
Figure 9. Sample XSL File for Importing (all code simplified for example) .....	15
Figure 10. Importing Another XSL File (all code simplified for example) .....	16
Figure 11. Web Page Produced Using XSL Import Files .....	17
Dynamic Interactive Sites .....	17
Figure 12. Logic, Content, and Style Separation in Cocoon Publishing Framework .....	18
Figure 13. XSL File Containing Logic Code (Cocoon XSP File, simplified for example) .....	19
Figure 14. Sample Dynamic HTML Page .....	20
Lessons Learned .....	21
Resources .....	22
Basic XML References .....	22
Other General XML References .....	22
DocBook .....	22
Tutorials on XML/XSL/XSLT .....	22
Newsletters, Lists, and Archives .....	22
Conversion Tools and Editors .....	23

---

# XML: A New Web Site Architecture

## ***A Better Way?***

As Web sites have grown in size, complexity, and prominence over the past five years, Web site management has become a growing concern for Webmasters, system administrators, and organizations as a whole. Unfortunately, the technology used to build most Web sites (HTML) is designed to produce individual Web pages easily, but does not provide a structure to easily maintain entire Web sites or manage the workflow involved in Web site production and maintenance.

Like many organizations, the Center for Technology in Government faced critical issues as its Web site matured over a five-year span from a simple location for posting reports and project results to a highly complex site with over 1,300 Web pages, thousands of hyperlinks, multiple navigation and search routes, interactive applications, and ongoing updates. The Web site also began to assume a more prominent status as a primary communication and outreach tool for the organization, so its performance, appearance, and timeliness became of greater concern. The question we confronted was not unusual:

*How could we continue to manage a Web site that was continually getting larger and more complex without just throwing more money and resources into it?*

There had to be a better way.

## ***Defining the Problem***

The question could be stated simply, but it was a big question. In looking for a solution, we needed to develop a better understanding of the problems involved. We identified three problems:

- Some of the tools and technologies upon which Web sites are based have built-in obstructions to effective ongoing development and maintenance.
- Web technologies are constantly evolving, so it's difficult to determine the best solution from an array of promising advances.
- Web site management is not just a technological challenge solvable with technological answers; it's an organizational and workflow challenge as well. Many Web site problems arise as workflow bottlenecks, so solutions must address workflow throughout the organization.

---

## **Partial Solutions**

In regard to the first two problems we identified, there was good news. HTML, the building block of Web pages, had evolved and continues to evolve as a more manageable and flexible tool for handling larger Web sites. Basic advances in the HTML specification such as the use of Cascading Style Sheets (CSS) to separate style from content marked a significant improvement in the ability to manage a Web site. With each new release, most HTML software packages added more site management and template features that enhanced the ability of Web developers to maintain consistency and propagate changes throughout a site.

However, we found that these developments did not really address the underlying problems of workflow and structure. They alleviated some difficulties, but still operated within the structural restrictions of the HTML page in which the content is never fully separated from the style. It works fine if you choose to involve the Web team (someone with HTML and Web scripting knowledge) in nearly all aspects of production, design, and delivery, and you create your Web site content for only one medium (the computer monitor Web page). But that is seldom, if ever, the case.

## **Addressing the Root Problems**

So was there an answer that addressed the root workflow problems and still offered a viable alternative to the status quo? Fortunately, XML (eXtensible Markup Language) seemed to address precisely these problems.

But what is XML? Here's a definition from the W3C Web site ([www.w3c.org](http://www.w3c.org)):

*XML is a set of rules (you may also think of them as guidelines or conventions) for designing text formats that let you structure your data. XML is not a programming language, and you don't have to be a programmer to use it or learn it. XML makes it easy for a computer to generate data, read data, and ensure that the data structure is unambiguous.*

“Structure your data.” That sounded like a step in the right direction. But let's stop for a moment to examine what “structured data” means within an XML context. Put more simply:

*XML is a markup language for documents containing structured information.*

“Markup” means that all text within a document is enclosed within tags that describe that content. Most people are familiar with markup through HTML where an `<h1></h1>` tag indicates a first-level heading and `<p></p>` indicates a new paragraph. However, notice that those two sample HTML tags do not really describe the data enclosed within them. They identify the enclosed data as headings or paragraphs, but do not describe the data itself. Is it a chapter heading? A paragraph within a list? While it tells a Web browser (such as Netscape or Internet Explorer) how to display the data, it does not define the type of data it contains. The same `<h1></h1>` tag can be used for the title of a book, the author, and the chapter titles. These are three very different types of data, but nothing within the HTML tags differentiates them.

---

XML employs markup in a fundamentally different way. If used properly, XML tags do not just identify data, they describe it. For example, a book title would be tagged with a `<title></title>` tag within a `<book></book>` tag. The author would be tagged within a `<author></author>` tag and even more precisely with `<firstname></firstname><lastname></lastname>` tags. The chapter title would be tagged with a `<title></title>` tag within a `<chapter></chapter>` tag. Notice that unlike HTML, the XML tags do not contain any information about how the data will be displayed, thus making it possible to separate content from format. XML describes the data more thoroughly in a richly structured document. Figure 1 provides a simple example of a structured XML document. Notice that the example contains no formatting information; that will be provided in a linked stylesheet file that contains instructions on how to display the XML content.

**Figure 1. Sample XML file (all code simplified for example)**

```
<?xml version="1.0" ?>
<!-- Specify the media type and the corresponding stylesheet -->

<?xml-stylesheet href="gateways_ss.xml" type="text/xml" media="netscape"?>
<?xml-stylesheet href="gateways_ss_ie.xml" type="text/xml"
media="explorer"?>
<?xml-stylesheet href="gateways_ss_lynx.xml" type="text/xml" media="lynx"?>

<book>
  <title>Opening Gateways:</title>
  <title>A Practical Guide for</title>
  <title>Designing Electronic Records Access Programs</title>

  <bookinfo>
    <authorgroup>
      <author>
        <firstname>Theresa A.</firstname>
        <lastname>Pardo</lastname>
      </author>
      <author>
        <firstname>Sharon S.</firstname>
        <lastname>Dawes</lastname>
      </author>
      <author>
        <firstname>Anthony M.</firstname>
        <lastname>Cresswell</lastname>
      </author>
    </authorgroup>

    <pubdate>December 2000</pubdate>

    <address>
      Center for Technology in Government
      University at Albany, SUNY
      <street>1535 Western Avenue</street>
      <city>Albany</city>
      <state>NY</state>
      <postcode>12203</postcode>
      <phone>(518) 442-3892</phone>
      <fax>(518) 442-3886</fax>
      <email>info@ctg.albany.edu</email>
      <otheraddr>http://www.ctg.albany.edu</otheraddr>
    </address>

    <copyright>
      <year>2002</year>
      <year>2000</year>
      <holder>Center for Technology in Government</holder>
    </copyright>

    <legalnotice>
      <para>The Center grants permission to reprint this document
      provided this cover page is included.</para>
    </legalnotice>

    <contractsponsor>This material is based upon work supported in
    part by the National Historical Publications and Records Com
    mission under Grant No. 98027.</contractsponsor>

  </bookinfo>
</book>
```

---

Now that you have a nicely structured document, how can you display it if there's nothing in the XML file to do that? Well, actually there is. As stated previously in the definition, XML is not a programming language; it's a set of rules for designing text formats to structure data. And XML is not just one thing; it is defined by several related specifications, including:

- **XML** (eXtensible Markup Language) which defines the syntax of XML (as seen in the sample in Figure 1)
- **XSL** (eXtensible Stylesheet Language) which is a language for expressing stylesheets. An XSL stylesheet is a file that describes how to display an XML document of a given type. It consists of three parts:
  - **XSLT** (Transformations), a language for transforming one XML document into another. It is used, for example, to generate HTML web pages from XML data.
  - **XPath**, (XML Path Language), a language used by XSLT to access or refer to parts of an XML document.
  - **XSL-FO** (Formatting Objects), an XML vocabulary for specifying formatting semantics and advanced styling features. It is used to produce a PDF document from an XML file, for example.

In short, XSL enables you to take your structured XML data and selectively format it for your various presentation possibilities.

Figure 2 contains a sample XSL file (simplified) that takes the content from the sample XML file in Figure 1 and prepares it for presentation as an HTML page, as shown in Figure 3. Note how the XSL file references the tags from the XML file (title, author, year, etc.) to retrieve the actual content and gives instructions on how to present the content within those XML tags as an HTML page (<h2 align="center">).

Now we have one place to create, control, and maintain our content (XML file) and another mechanism to present that content (XSL file). We have effectively separated content from style and begun to address our root problem. By comparison, HTML tags contain both content and style information (such as <h2></h2> which identifies a second-level heading and defines how to present it) or contain just style information (such as <strong></strong> which instructs a browser to show enclosed text as bold).

**Figure 2. Sample XSL File (all code simplified for example)**

```
<?xml version="1.0"?>

<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">

  <!-- Specify the XHTML layout for the root tag -->

  <xsl:template match="book">
    <html>
      <head>
      </head>
      <body>

        <!-- Specify to look for the styling for title element-->

        <xsl:apply-templates select="title"/>

        <br/><br/>

        <xsl:apply-templates select="author"/>
        <xsl:apply-templates select="year"/>

        <br/><br/>

        <p align="center">
          <xsl:apply-templates select="etc"/>
        </p>

        <xsl:apply-templates select="footer1"/>
        <br/>

        <xsl:apply-templates select="footer2"/>
        </body>
      </html>

    </xsl:template>

    <!-- Specify the xhtml style for the 'title' in xml file -->
    <xsl:template match="title">
      <h2 align="center"><xsl:value-of select="."/></h2>
    </xsl:template>

    <!-- Specify the xhtml style for the other templates -->

    <xsl:template match="author">
      <h3 align="center"><xsl:value-of select="."/></h3>
    </xsl:template>

    <xsl:template match="year">
      <h4 align="center"><xsl:value-of select="."/></h4>
    </xsl:template>

    <xsl:template match="etc">
      <font align="center" size="2pt"><xsl:value-of select="."/><br/></font>
    </xsl:template>

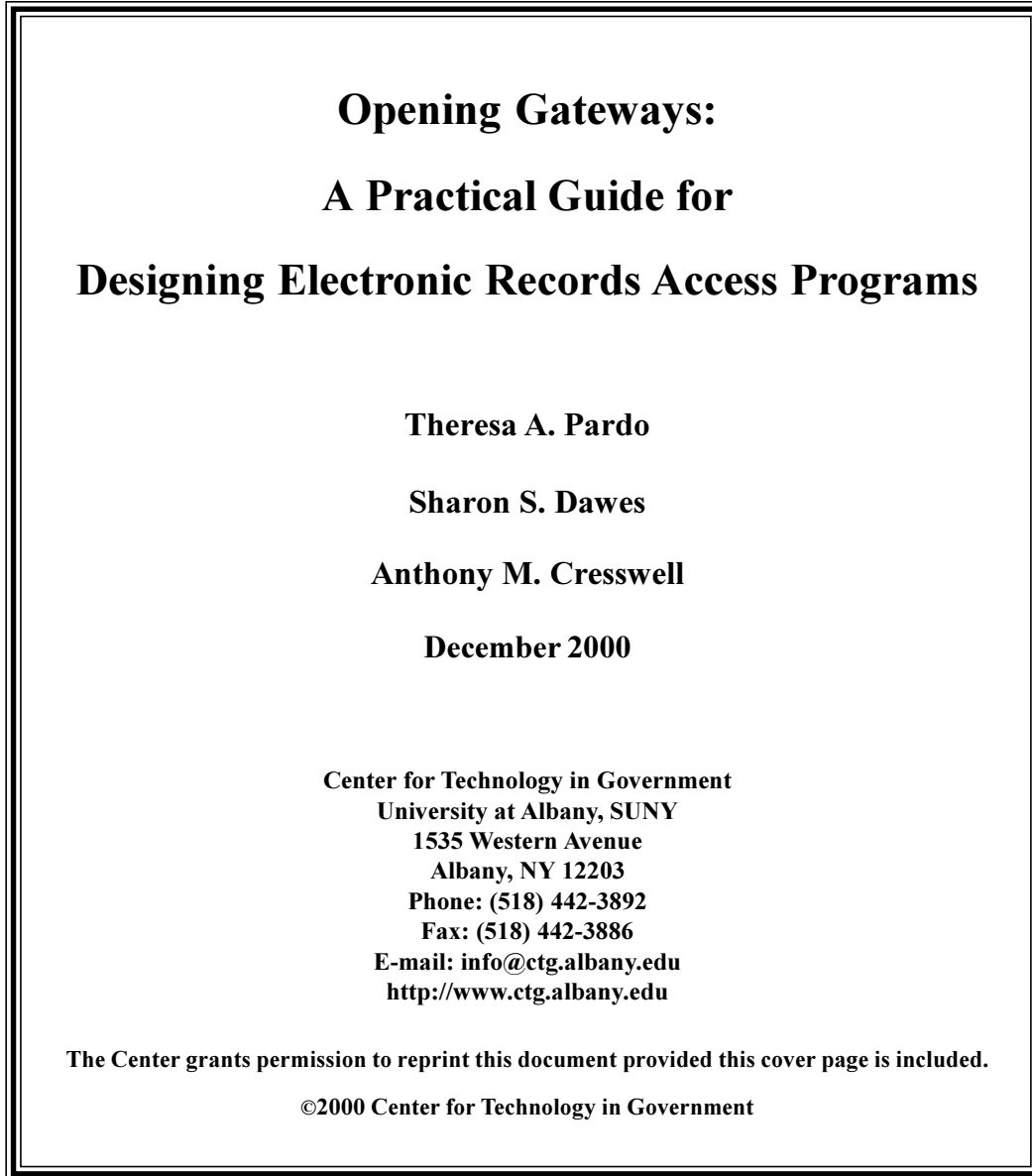
    <xsl:template match="footer1">
      <h5 align="center"><xsl:value-of select="."/></h5>
    </xsl:template>

    <xsl:template match="footer2">
      <h5 align="center"><xsl:value-of select="."/></h5>
    </xsl:template>

  </xsl:stylesheet>
```



**Figure 3. Formatted Page Produced from Sample XML File using XSL**



### ***Workflow Impact***

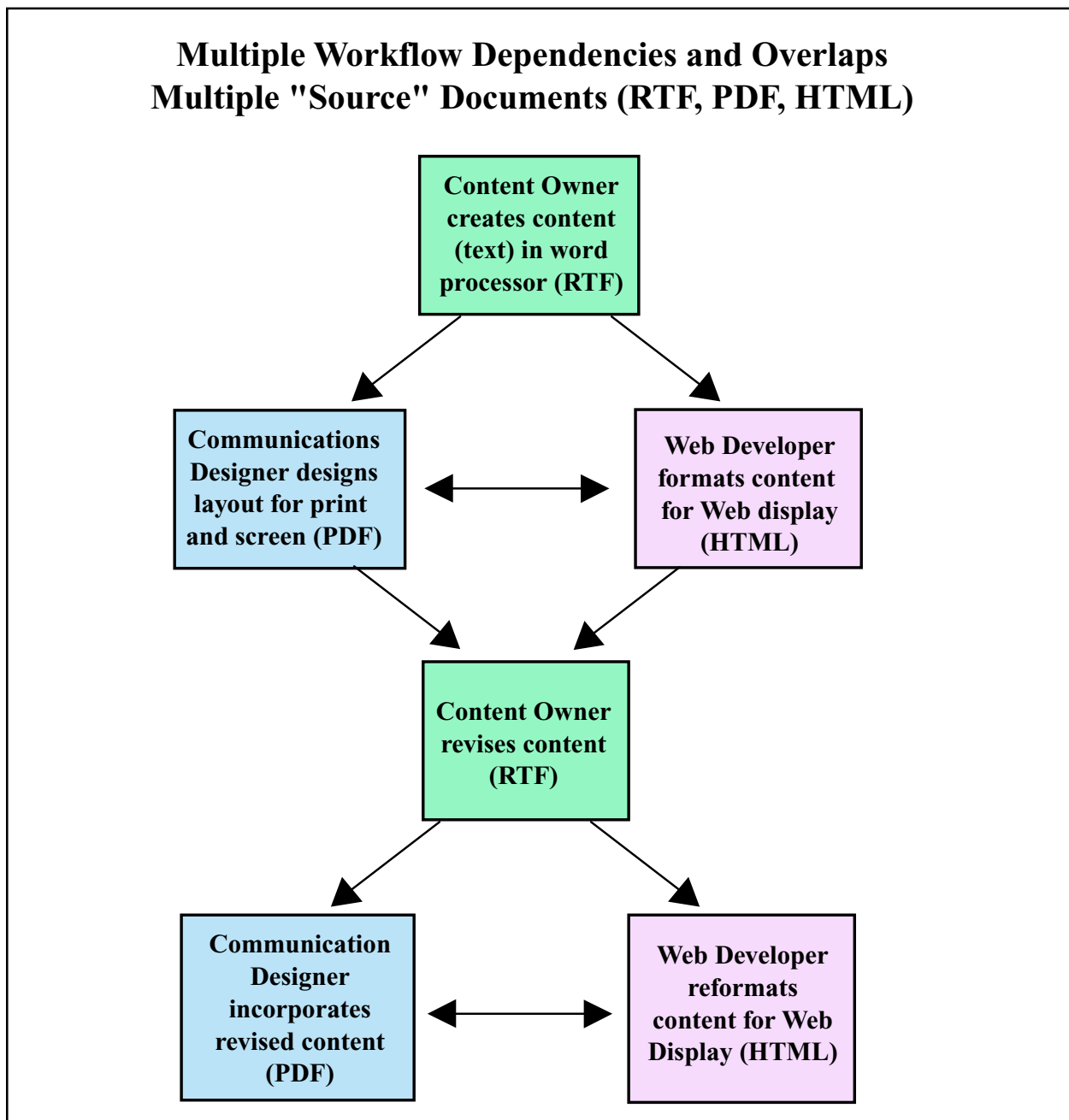
The separation of content from style enabled us to effectively separate the functions associated with content and style. Now the content owners (subject matter experts, project managers, communications staff) could work on their content without the intermediary of a Webmaster or HTML coder. In other words, content would not be developed once in a word processing format and then redeveloped again in an HTML format and then likely redeveloped again and again for various HTML pages or browsers.

The content could now reside in one place only: the XML file. The XSL stylesheets determine how the content is presented in the various Web pages and browsers on which the content will be displayed. And the presentation is not restricted to Web pages. The XSL stylesheets can

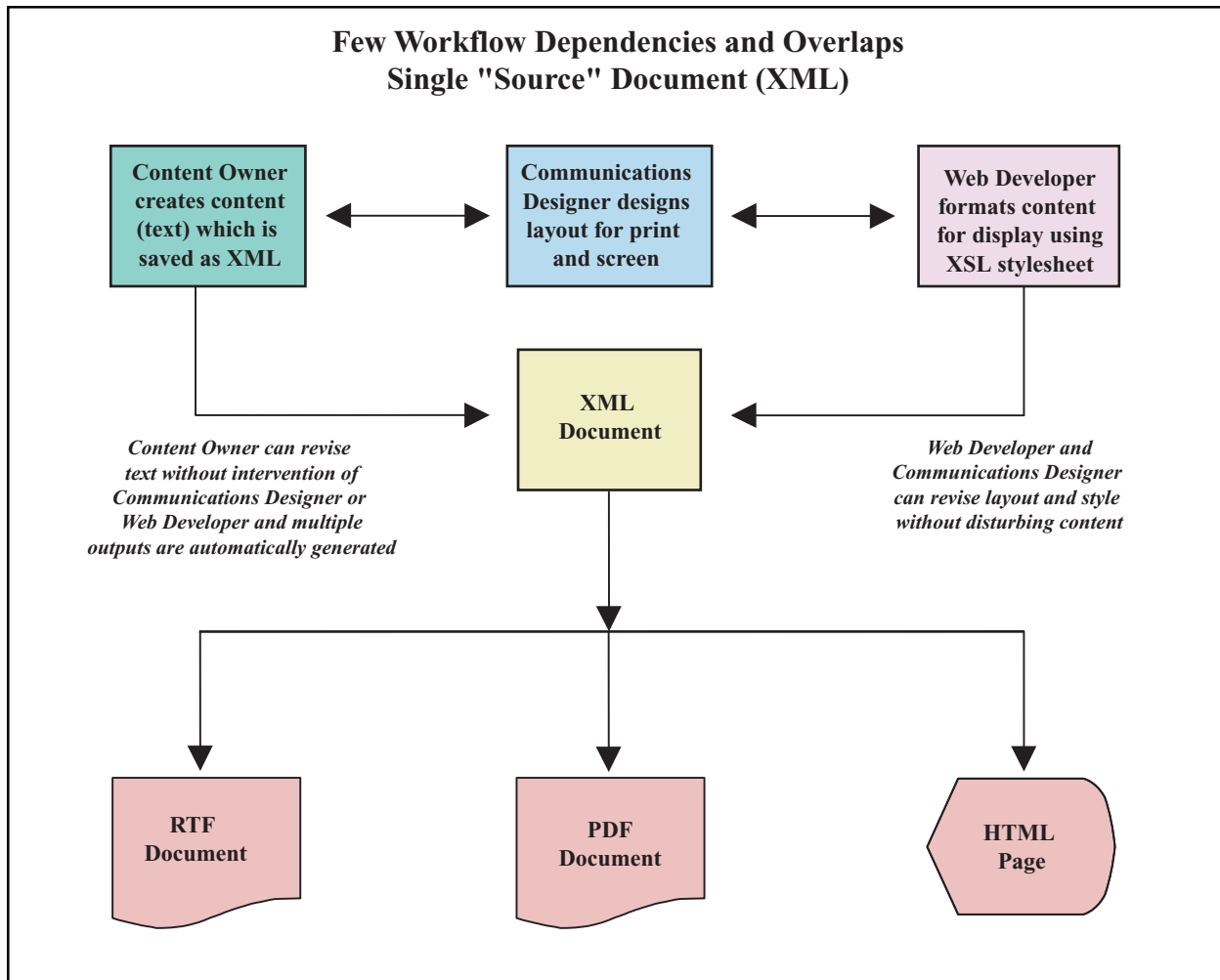
transform the XML file into HTML, PDF, RTF, WML – a variety of options – all working from a single XML file.

Figures 4 and 5 illustrate the workflow impacts in moving from an HTML-based Web architecture to an XML-based architecture. One of the advantages of using XML, as seen in these diagrams, is that it eliminates many dependencies and redundancies in the workflow, allows teams to work independently, and maintains a single source document (XML) with multiple deliveries (HTML, RTF, PDF, etc.).

**Figure 4. Workflow (Simplified) in HTML-based Web Site Management**



**Figure 5. Workflow (Simplified) in XML-based Web Site Management**



### **Working with XML**

Recognizing that XML offers solutions to Web site architecture problems does not by itself answer questions of how to work with XML or whether the use of XML is even realistic today. We all live in the real world, and when dealing with Web sites, need to deal with real issues of multiple browsers and display devices and connections. For today's world (2002), this means that much of the XML processing must be handled on the Web server, not on the desktop or client device. The server "translates" the XML files and sends the appropriate HTML pages to the client browser.

Although newer web browsers, such as Internet Explorer 6 and Netscape 7, offer some XML/XSL support, the consistency and depth of support for the various XML specifications within client browsers is unpredictable and unreliable. Therefore, we need to perform our XML processing on the server. In addition, our Web site, like most Web sites today, also has database and programming elements so we wanted to integrate these elements within the XML structure. For these reasons we adopted the Apache Cocoon XML publishing framework for developing and delivering our XML files. (For more information on Cocoon, which is an open-source,

---

server-side environment for XML processing that supports full integration of database and programming logic, visit [xml.apache.org/cocoon/](http://xml.apache.org/cocoon/)).

On the development side, we encountered a similar situation in that XML software tools are rather primitive compared to HTML tools. Some XML editors, such as XML Spy ([www.xmlspy.com](http://www.xmlspy.com)) offer useful tools for developers, but are not really designed for other users such as content owners and subject matter experts. In other words, a Web developer could use it to create XML and XSL files, but content owners could not easily use it now for editing their original document and saving it as XML. However, these software packages are heading in that direction and becoming more user-friendly with time.

### **Converting Our Site**

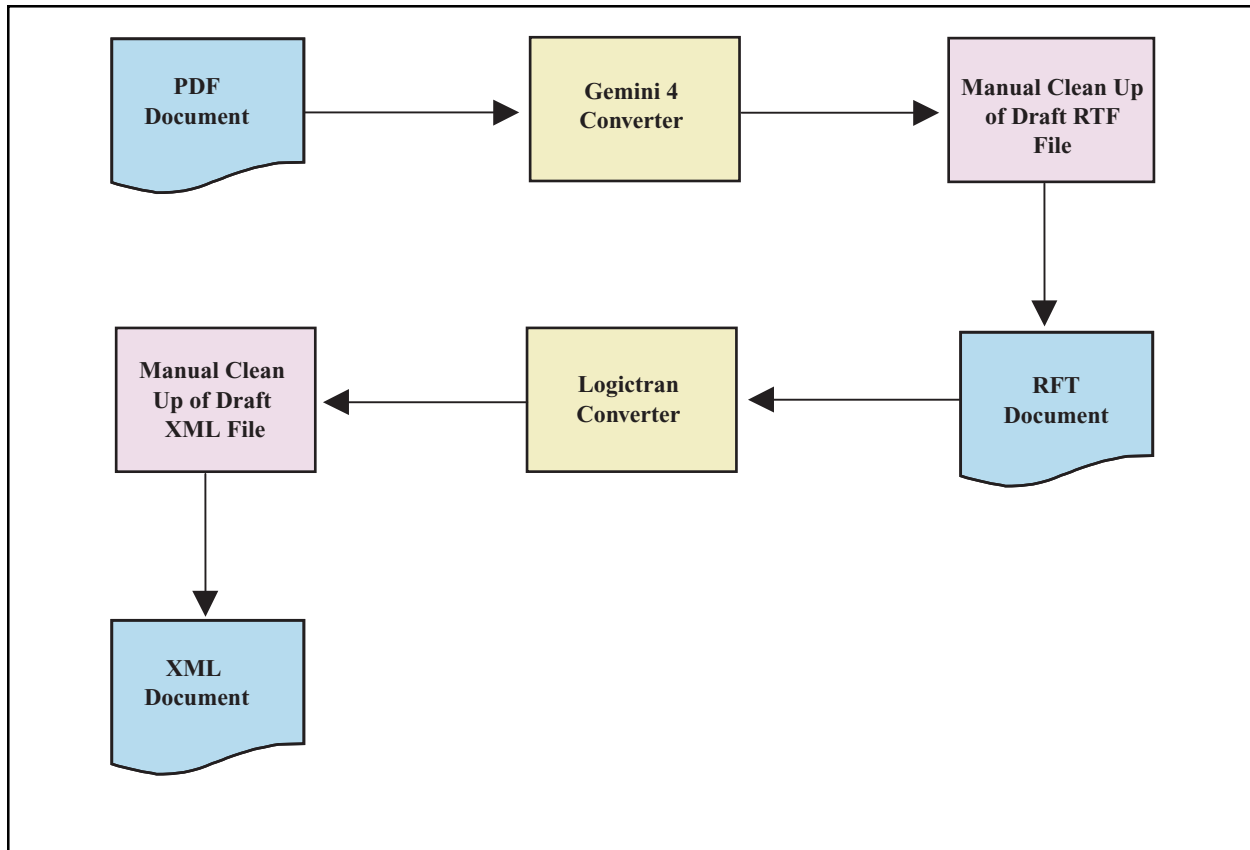
Once we decided what to do (convert to XML) and how to do it (use the Cocoon publishing framework), we were ready to begin the actual task of doing it. Since much of our Web site consists of documents (reports, project results, presentations), we began by converting one of these documents, *Opening Gateways: A Practical Guide for Designing Electronic Records Access Programs*, into an XML format. We used this 50-page guide as a prototype to test the viability of converting our Web site while it enabled us to learn XML and the Cocoon environment.

The first step was to convert the existing document. Like most of our publications, the final version of the *Gateways Guide* was in a PDF format. Our goal was to convert this PDF into an XML format that could serve as our final version, single-source file. We also wanted to automate this process as much as possible to reduce the time involved and ensure consistency in the results. To reach this goal (from PDF to RTF to XML), we used a four-step process with two software conversion programs and two “clean-ups” after each conversion as illustrated in Figure 6.

Our resulting XML document conformed to the DocBook standard which is a widely used document type definition standard containing a popular set of tags for describing books and articles (such as <book>, <chapter>, <title>, etc.). Within XML, it’s important to use standard, widely accepted markup tags to describe your data so that you can use and share this data over time and in a variety of applications. While nothing within XML prohibits you from creating your own markup tags, it is not good practice because it potentially isolates your content and limits the flexibility of stylesheets to transform and present that content. If like items (such as chapter titles) are referenced in like ways, then one XSL stylesheet, for example, could transform an unlimited number of different XML documents.

This conversion process worked very well for us and allowed us to convert a document within a few hours. This does not mean that we recommend the same conversion process and tools in all cases. Every environment has different needs. Fortunately, several conversion tools and methods are available for a variety of situations. (See our list of references at the end of this document for additional information on conversion tools.)

**Figure 6. Transforming Documents from PDF to RTF to XML**



### ***The Advantage of an XML Document***

Why go to the trouble of converting the document from PDF to XML if it's ultimately going to be presented on the Web site as a PDF document? The advantage of converting the document to XML can be seen in Figure 5. When the final document is an XML document, you can present it as PDF, HTML, RTF and a variety of other formats including mobile and voice displays. In our case, that is a major advantage.

Our Web site contains approximately 50 documents ranging from 50 to 100 pages. Most of these are in a PDF final format and not in HTML at all. Just to produce these same documents in an HTML format would nearly double the size of our Web site in files and pages. Plus, it would double the complexity of maintenance since we would have at least two final versions (one in PDF and one in HTML). In many cases, we would have three final versions (a Word document as well) and frequently a PageMaker® or FrameMaker® version for printed publications.

Looking back, Figure 4 illustrates the impact of these multiple versions on workflow. For example, every time the content changes in the Word document, a web developer has to edit every HTML page impacted by the change and a new PDF has to be produced. Something as simple as changing the title of the guide could involve changes to 50 different HTML files and manual proofreading of documents (Word, HTML, PDF) to verify consistency.

---

The complexity increases when browser incompatibilities are considered. Take a simple example of providing a text-only version of the document for users who do not use graphical browsers. Not everyone uses Internet Explorer, Netscape, Opera or Mozilla; some people use Lynx or other non-graphical browsers. Now we need 50 HTML pages for the graphical version, and another 50 for the text-only version. Furthermore, if we decide to use browser-specific features (IE6 and Netscape 4 for example), we have again increased the number of HTML pages on our site.

XML alleviates these problems by:

- Creating a single source for the final version of your content (it's in one XML file as shown in Figure 5)
- Using multiple XSL files to selectively transform the XML document into the appropriate output format (HTML for graphic and text-only browsers, PDF, etc.)

Rather than having multiple source files and hundreds of HTML files, a document like the 50-page *Gateways Guide* would have one source file and perhaps 5–10 XSL files to produce the multiple output formats. As content changes occur, the change is made in the single XML file and then automatically and immediately propagated to the various output formats. A Web developer does not have to modify dozens or hundreds of HTML pages; no one has to generate a new PDF using Adobe® Acrobat®; and no one has to proofread all the various formats to ensure the change was applied consistently.

Cocoon offers additional features to further ease these maintenance issues. For example, one-line parameters identify different browsers and automatically direct users to different stylesheets appropriate for them. Instead of 50 different HTML pages for each type of browser, you have one line and one additional XSL file. (The next section examines these Cocoon features and XSL files more closely.)

Unlike the HTML-based architecture which suffers from increased maintenance impacts as the Web site grows in size and complexity, an XML-based architecture actually levels off. There is a limited impact on maintenance as the site gets bigger and more complex.

The ease in managing the content comes from the basic property of XML that provides a total separation of content (source) and style (output). The content or data resides in a single XML document and different XSL stylesheets present that data differently.

---

## Transforming Content with XSL

Let's look at the XSL stylesheet to see how it transforms and presents an XML document. The first thing to understand about an XSL file is that it does not contain any data. It contains instructions on how to select and present the data from an XML file. But how does an XML file know which XSL stylesheet to use? It's quite simple actually. In the opening lines of an XML document, you include a processing instruction that links to the stylesheet. Figure 7 displays these processing instructions (<?xml-stylesheet ...?>) for different browsers. In this example, the XML file would use the "gateways\_ss.xml," "gateways\_ss\_ie.xml," or "gateways\_ss\_lynx.xml" stylesheet depending on the client browser. The "media=" parameter at the end of each line is a Cocoon feature that performs automatic browser detection and directs users to the appropriate stylesheet for their browsers.

**Figure 7. Connecting XML and XSL Files**

```
<?xml version="1.0" ?>
<!-- Specify the media type and the corresponding stylesheet

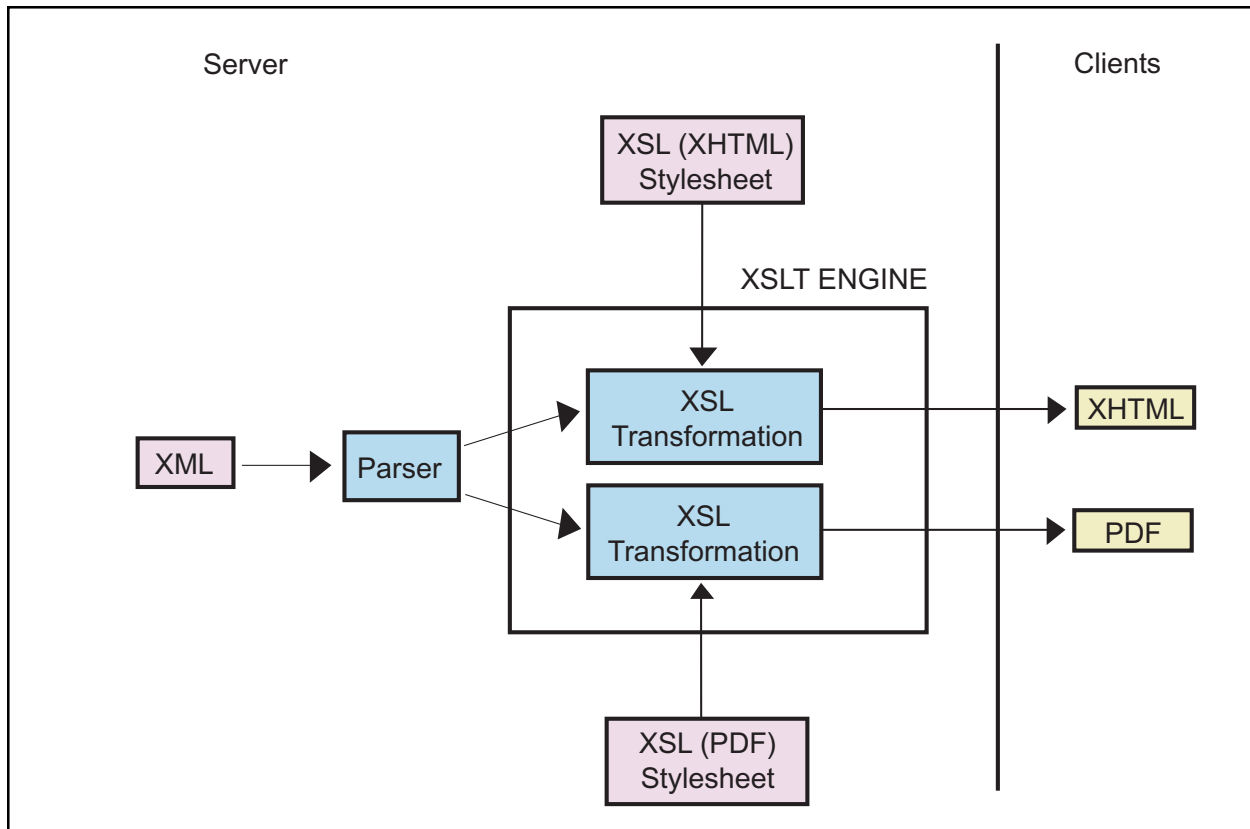
<?xml-stylesheet href="gateways_ss.xml" type="text/xsl" media="netscape"?>
<?xml-stylesheet href="gateways_ss_ie.xml" type="text/xsl" media="explorer"?>
<?xml-stylesheet href="gateways_ss.xml" type="text/xsl" media="mozilla5"?>
<?xml-stylesheet href="gateways_ss_lynx.xml" type="text/xsl" media="lynx"?>
<?xml-stylesheet href="gateways_ss.xml" type="text/xsl"?>
```

The XSL file then uses "templates" to identify the content to select from the XML file and specify how to format that content. A "template" corresponds to the tags in the XML document. For example, if you look back to the sample XSL file in Figure 2, you'll see it has templates for "book," "title," "author" and so on that correspond to the tags in the original XML document. The XSL file is selecting the content within those tags and formatting it with the HTML tags within its "<xsl:template match>" tags. This XSL file is styling the content for display as an HTML page in a browser. Different browser types may have different stylesheets if you choose to design for different browsers. A PDF output would also have a different stylesheet. But all the stylesheets use the same source XML document.

This significantly simplifies content management because it enables you to make changes to only one content file and make no changes to the stylesheets because the content is totally separated from the presentation.

Figure 8 illustrates how these XML/XSL transformations occur within the Cocoon publishing framework. An XML file is processed on the server by a "parser" that analyzes and validates the XML tags. It then uses the XSL stylesheets to transform the XML document into the appropriate output format for the clients (PDF or HTML).

**Figure 8. Transforming XML Documents into XHTML and PDF**



### ***Maintaining Style Consistency***

Another advantage with the XML-based architecture is easier maintenance of consistency throughout the site. Repeated HTML elements such as the top banner, left navigation, and footer can be stored in separate stylesheets and imported into other stylesheets for presentation throughout the Web site. Changes to the banner or footer are made in only one file and propagated throughout all pages on the site. Figure 9 contains an XSL file for the top banner of the *Gateways Guide* pages.



**Figure 9. Sample XSL File for Importing (all code simplified for example)**

```
<?xml version="1.0"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="banner">
<!-- HTML code for the CTG banner -->

    <!-- Begin Banner Section -->
    <div align="left">
    <table border="0" cellpadding="0" cellspacing="0" width="705">
    <tr>
    <td align="left" valign="top" width="57" height="75">
    <a href="http://www.ctg.albany.edu/">
    </img>
    </a>
    </td>
    <td align="left" valign="middle" width="658" class="name" >
    Center for Technology in Government<br/>
    <span class="univ">University at Albany, State University of New York</strong></
    span></td>
    </tr>
    </table>
    </div>
    <!-- End Banner Section -->

    <!-- Begin Menu Bar Section -->
    <table border="0" cellpadding="0" cellspacing="0" width="705">
    <tr>
    <td width="705" height="20" colspan="2" align="center" valign="top"
    bgcolor="#2f538a">
    <a href="http://www.ctg.albany.edu/whatsnew/whatsmn.html">What's New</a>
    <a href="http://www.ctg.albany.edu/aboutctg/aboutmn.html">About</a>
    <a href="http://www.ctg.albany.edu/projects/projmain.html">Projects</a>
    <a href="http://www.ctg.albany.edu/research/researchm.html">Research</a>
    <a href="http://www.ctg.albany.edu/resources/rptwplst.html">Reports</a>
    <a href="http://www.ctg.albany.edu/resources/tool.html">CTG Toolbox </a>
    <a href="http://www.ctg.albany.edu/partners/partnrnm.html">Partners</a>
    <a href="http://www.ctg.albany.edu/education/edumn.html">Education</a>

    <a href=http://www.ctg.albany.edu/resources/resorcmm.html>Resources</a>
    </td>
    </tr>
    </table>

</xsl:template>

</xsl:stylesheet>
```

Importing the file into another stylesheet is also quite simple, as illustrated in Figure 10. You use an XSL import tag to specify the file to import (such as “<xsl:import href=“banner.xml”/>”). Then, you use an apply-imports tag (such as the “<xsl:apply-imports select=“banner”/>”) where you want the imported file to appear. Now your banner exists in a single file used by the entire site. If you change your banner then you only change that one file, and all pages display the same banner. That helps us to maintain consistency throughout our site.

**Figure 10. Importing Another XSL File (all code simplified for example)**

```
<!-- import external xsl files for banner -->

<xsl:import href="banner.xsl"/>

<!-- Start the HTML Layout here -->

<xsl:template match="book">
  <html>
    <head>
      <title><xsl:value-of select="@bookname"/>:<xsl:value-of
select="chaptitle"/></title>

      <link rel="stylesheet" type="text/css" href="YOUR_CSS.css"
title="Style"/>
    </head>
    <body>

      <table width="705" cellspacing="0" cellpadding="0">

        <!-- Start the the Top section here -->
        <tr width="705">
          <td width="705">
            <!--PULL IN THE IMPORTED BANNER STYLESHEET HERE-->
            <xsl:apply-imports select="banner"/>
          </td>
        </tr>
        <!-- End the the Top section here -->
      </table>
    </body>
  </html>

  <!-- Use the imported Banner stylesheet for styling-->
  <xsl:template match="banner">
    <xsl:apply-imports/>
  </xsl:template>

</xsl:stylesheet>
```

The Web page produced for our sample *Gateways Guide* using the XML and XSL files described in the previous pages is shown in Figure 11.

Figure 11. Web Page Produced Using XSL Import Files

The screenshot shows a web page with a header for the Center for Technology in Government at the University at Albany. A navigation bar contains links like 'What's New', 'About CTG', 'Projects', 'Research', 'Publications', 'CTG Toolbox', 'Partners', 'Education', 'Resources', and 'CTG Homepages'. Below this is a secondary navigation bar with links for 'Opening Gateways Home', 'Gateways Project Home', 'Interactive Tools', 'PDF Version', 'Text Only Version', and 'Email Document'. The main content area features a large title 'Opening Gateways: A Practical Guide for Designing Electronic Records Access Programs' by Theresa A. Pardo, Sharon S. Dawes, and Anthony M. Cresswell, Second Edition January 2002. Contact information for the Center for Technology in Government is provided, including address, phone, fax, email, and website. A sidebar on the left contains sections for 'Acknowledgements', 'Introduction', and 'Assessment Tool', each with a list of links. A footer note states: 'The Center grants permission to reprint this document provided this cover'.

### Dynamic Interactive Sites

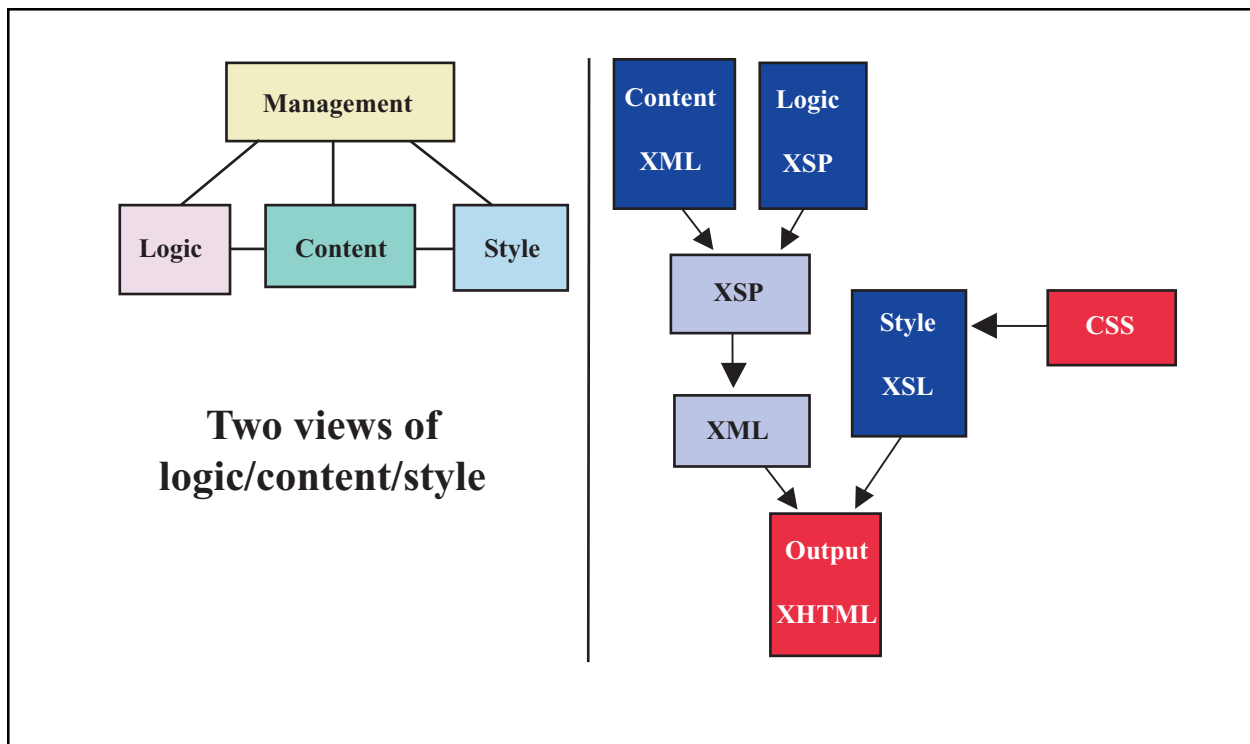
The examples used so far have focused on separation of content and style in static pages; that is, HTML pages with predefined text and images. This type of document lends itself to an XML framework perfectly due to its inherent structure, stability, and standardization. A publication such as the *Gateways Guide* adheres to a hierarchy with chapters and sections and paragraphs. For that type of Web site, XML is a great match.

However, few Web sites contain only that type of content. Most Web sites today have some kind of dynamic and interactive content involving external files or databases and transactions. Even simple forms involve some level of interaction.

So now we have to consider how to handle this situation where content is not restricted to text in an XML file, but includes data in a relational database, and content selection involves not just XSL transformations, but programming logic. Can we incorporate different data structures and processing algorithms into our XML framework?

As Figure 12 illustrates, Cocoon does offer a structure for handling this situation by not only separating content from style, but from logic as well. Within Cocoon, the separation of logic is handled in a special type of file called an “XSP” file.

**Figure 12. Logic, Content, and Style Separation in Cocoon Publishing Framework**



This structure allows you to manage the process efficiently by segregating duties. An application programmer, for example, can work on the processing logic without worrying about or, more importantly, interfering with the style or content elements. In fact, it enables parallel development because logic, content, and style are worked on simultaneously and independently and then integrated in the final output.

Figure 13 shows an XSP file which contains only logic (in this case, Java and SQL code), no content or style information. The XSP file is connected to its XML and XSL files in exactly the same way that the XML file was connected to the XSL file: through processing instructions at the start of the file that link it to the other files. In fact, an XSP file has a .xsl file extension so the XML file links to the XSP file which links to the XSL stylesheets.

**Figure 13. XSL File Containing Logic Code  
(Cocoon XSP File, simplified for example)**

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xsp="http://www.apache.org/1999/XSP/Core"
  xmlns:esql="http://apache.org/cocoon/SQL/v2">

<!--COMMENT Code example simplified for example -->

<xsl:template match="page">
<xsl:processing-instruction name="xml-stylesheet">href="registershow.xsl" type="text/xsl"
</xsl:processing-instruction>

<!-- COMMENT Defines the query string for the database -->
<xsp:logic><![CDATA[
String username,programe,password,t;
private String assemblyQuery()
{
t="select * from XXXXXXXX where YYYYYY='"+username+"'";
return t;
}  ]]>
</xsp:logic>

<!-- COMMENT verifies valid entry and redirects or allows in-->
<xsp:logic><![CDATA[
if (request.getParameter("username")==null) {
response.sendRedirect("signon.xml");
}]]>
else {![CDATA[
username=request.getParameter("username").replace('\','');
]]>

<!-- COMMENT Database connection and query -->
<esql:connection>
  <esql:driver>org.gjt.mm.mysql.Driver</esql:driver>
  <esql:dburl>jdbc:mysql://XXXXXXXX/XXXXXXXX</esql:dburl>
  <esql:username>XXXXXX</esql:username>
  <esql:password>XXXXXX</esql:password>
  <esql:execute-query>
    <esql:query><xsp:expr>assemblyQuery()</xsp:expr></esql:query>
    <esql:results>
      <esql:row-results>
        <user>
          <username><esql:get-string column="username"/></username>
          <programe><esql:get-string column="programe"/></programe>
        </user>
      </esql:row-results>
    </esql:results>
    <esql:no-results>
      <norecords/>
    </esql:no-results>
    <esql:error-results>
      <dbproblem/>
    </esql:error-results>
  </esql:execute-query>
</esql:connection>

<!-- COMMENT Closes out the big else condition from the redirect if -->
}
</xsp:logic>

<!-- COMMENT Closes out the template and the stylesheet -->
</xsl:template>
</xsl:stylesheet>
```

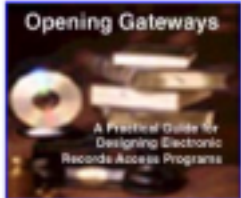
This content/logic/style structure enabled us to develop an interactive component of the *Gateways Guide* in which users using online forms could register and create online worksheets that employ the principles and practices described in the guide. Most importantly, the logic for this system (involving a MySQL database, SQL, Java, and JavaScript coding) was developed independently by application programmers using XSP files that were integrated into the existing XML/XSL framework established for the guide. The result as seen in Figure 14 was an interactive, dynamic application that not only had the same look and feel as the *Gateways Guide* but actually used the same XML source documents and XSL stylesheets. In other words, content and style was not duplicated or re-created for this new application; it was simply re-used.

Figure 14. Sample Dynamic HTML Page

 **Center for Technology in Government**  
University at Albany, State University of New York

[What's New](#) | [About CTG](#) | [Projects](#) | [Research](#) | [Publications](#) | [CTG Toolbox](#) | [Partners](#) | [Education](#) | [Resources](#) | [CTG Homepage](#)

**Opening Gateways** | [Opening Gateways Home](#) | [Gateways Project Home](#) | [Text Only Version](#)

 **Opening Gateways**  
A Practical Guide for  
Designing Electronic  
Records Access Programs

[Previous Screen](#)

[Case Study](#)

[Tutorial](#)

## Sign on to use Gateway Tools

You must sign on to use the online, interactive Gateway tools. If you have previously registered, your signon will retrieve the Program Plan name(s) assigned to you. You may then access an existing Program Plan or create a new Program Plan.

If this is your initial registration, you will be given an option to access an existing Program Plan or create a new Program Plan.

Click the **Tutorial** link at the left for a brief step-by-step walk-through on using the system.

**E-mail Address**   
(your e-mail address provides a unique user name)

[What's New](#) | [About CTG](#) | [Projects](#) | [Research](#) | [Publications](#) | [CTG Toolbox](#) | [Partners](#) | [Education Program](#) | [CTG Homepage](#)  
[Resources](#) | [Table of Contents](#) | [Privacy Statement](#) | [Contact Us](#) | [Search Our Site](#) | [CTG Web News](#)

---

## **Lessons Learned**

From our initial question of was there a better way to manage a Web site, our exploration of XML and Cocoon delivered a resounding “yes.” XML not only alleviated some immediate problems, but more importantly had a positive impact on fundamental workflow issues.

But we also realized that conversion of a Web site does not happen without some pain. The learning curve was probably the biggest impact we experienced. For our initial developers who were investigating XML and Cocoon for the first time with limited external resources and no in-house knowledge, we measured a three-month learning curve before they became fully productive.

Much of XML is similar to HTML, but much is different. The most significant difference is that XML is not “page-based” like HTML, so it requires a different conceptual understanding and approach. Whereas in HTML, you are basically creating pages, and the page is both your source and output; in XML you are structuring your content and designing different deliveries. It is a radically different way of thinking about your Web site. While some knowledge and skills are transferable from HTML to XML, the same knowledge and skills can also be detrimental to thinking within an XML framework. Even experienced HTML developers and programmers need to learn anew as they move into an XML-based architecture.

On the positive side, our three-month learning curve has sharply declined within the past year as external resources (books, Web sites, user groups) have increased and our in-house knowledge base has grown. Experienced HTML Web developers brought on board since our *Gateways Guide* prototype has been completed now experience a learning curve of three weeks (versus the original team’s three months).

---

# Resources

## Basic XML References

Cocoon [xml.apache.org/cocoon/index.html](http://xml.apache.org/cocoon/index.html)

Cocoon [xml.apache.org](http://xml.apache.org)

W3C Specifications [www.w3.org](http://www.w3.org)

O'Reilly's XML Site [www.xml.com](http://www.xml.com)

OASIS Site (General XML) [www.oasis-open.org](http://www.oasis-open.org)

XML Cover Pages Site (OASIS) [xml.coverpages.org](http://xml.coverpages.org)

IBM Developer Works [www-106.ibm.com/developerworks/xml](http://www-106.ibm.com/developerworks/xml)

Site Using Cocoon [www.gsa.gov/attachments/GSA\\_PUBLICATIONS/extpub/11-Ballif-Vock-Switzerland.htm](http://www.gsa.gov/attachments/GSA_PUBLICATIONS/extpub/11-Ballif-Vock-Switzerland.htm)

## Other General XML References

The XML FAQ [www.ucc.ie/xml](http://www.ucc.ie/xml)

XML Hack [www.xmlhack.com](http://www.xmlhack.com)

DevX [www.devx.com/xml](http://www.devx.com/xml)

XML 101 [www.xml101.com](http://www.xml101.com)

Dublin Core Metadata Initiative [www.dublincore.org](http://www.dublincore.org)

## DocBook

The DocBook standards [www.docbook.org](http://www.docbook.org)

Simple DocBook [www.docbook.org/xml/simple/index.html](http://www.docbook.org/xml/simple/index.html)

Docbook Slide and Website [sourceforge.net/projects/docbook](http://sourceforge.net/projects/docbook)

PDF Example using DocBook Slides [www.sun.com/software/xml/developers/slides-dtd](http://www.sun.com/software/xml/developers/slides-dtd)

Introduction to DocBook [www-106.ibm.com/developerworks/library/l-docbk.html](http://www-106.ibm.com/developerworks/library/l-docbk.html)

## Tutorials on XML/XSL/XSLT

Transforming XML: [www-106.ibm.com/developerworks/education/transforming-xml/index.html](http://www-106.ibm.com/developerworks/education/transforming-xml/index.html)

Introduction to XSLT, Part I: [www.webreview.com/2001/08\\_03/developers/index01.shtml](http://www.webreview.com/2001/08_03/developers/index01.shtml)

Introduction to XSLT, Part II: [www.webreview.com/2001/08\\_10/developers/index02.shtml](http://www.webreview.com/2001/08_10/developers/index02.shtml)

Introduction to XSLT, Part III: [www.webreview.com/2001/09\\_21/developers/index02.shtml](http://www.webreview.com/2001/09_21/developers/index02.shtml)

Introduction to XSLT, Part IV: [www.webreview.com/2001/10\\_29/developers/index01.shtml](http://www.webreview.com/2001/10_29/developers/index01.shtml)

Introduction to XSLT, Part V: [www.webreview.com/2001/11\\_26/developers/index01.shtml](http://www.webreview.com/2001/11_26/developers/index01.shtml)

Introduction to XSLT, Part VI: [www.webreview.com/2001/12\\_20/developers/index01.shtml](http://www.webreview.com/2001/12_20/developers/index01.shtml)

XSL Concepts and Practical Use: <http://www.nwalsh.com/docs/tutorials/xsl/xsl/slides.html>



---

## **Newsletters, Lists, and Archives**

XML Applications in Government (USGSA): [www.gsa.gov/intergov](http://www.gsa.gov/intergov)

Inside XML Solutions (discontinued): [www.elementkjournals.com](http://www.elementkjournals.com)

XML Developers: [www.xmldevelopernewsletter.com](http://www.xmldevelopernewsletter.com)

Monthly Archives for XML-DEV (OASIS): [lists.xml.org/archives/xml-dev](http://lists.xml.org/archives/xml-dev)

Mulberry Open Forum on XSL [www.mulberrytech.com/xsl/xsl-list](http://www.mulberrytech.com/xsl/xsl-list)

## **Conversion Tools and Editors**

General Reference on Conversion Tools: [www.xmlsoftware.com/convert](http://www.xmlsoftware.com/convert)

XMLPDF: [www.xmlpdf.com](http://www.xmlpdf.com)

RTF Formatting Kit: [www.schema.de/sitehtml/site-e/xmlnach0.htm](http://www.schema.de/sitehtml/site-e/xmlnach0.htm)

PDF to RTF Converter: [www.iceni.com/geminiSet.html](http://www.iceni.com/geminiSet.html)

RTF to XML Converter: [www.logictran.com](http://www.logictran.com)

XML Editor: [www.xmlspy.com](http://www.xmlspy.com)