Chapter 1

The Cocoon 2 Architecture

- The Challenges of Web Publishing
- The Challenges of Web Content Management
- Content Management Systems to the Rescue
- The Original Cocoon Project
- Architecture of the Cocoon 2 Framework

Solutions for web publishing abound and have been rapidly evolving to fill the needs of both developers and their customers. All of these solutions come down to systems for managing content and serving it up on the Web upon request in a format easily accessible to a variety of users.

This chapter introduces the Cocoon framework by defining the original design goals and architecture of Cocoon 1 and studying the areas where the framework fell short. The chapter also provides an overview of the Cocoon 2 architecture and lays the foundation for developing robust web applications using the framework. In the course of this chapter and the rest of the book, we highlight the benefits that the Cocoon framework brings to most web-publishing efforts and why most projects benefit from it.

So why Cocoon 2? Because it's a flexible, powerful solution based on XML—the language of the Web—and related technologies. It is part of the Apache Project, making it fully capable of managing content and presentation for large-scale web publishing.

To fully understand Cocoon's place in the scheme of things, however, we must take a brief look at the origins of the World Wide Web and the technologies that supported the adoption of the Web as the *de facto* standard for information repositories around the world. Ever since the mid-1990s, the popularity of the Web has helped to shape the evolution of strategies for business. This chapter emphasizes the associated problems that came about and the issues that have created challenges for technical and business organizations as they tried to adapt to the constantly changing needs that were created by the Web.

Finally, we take a look at the challenges of developing and maintaining web applications and the impact those challenges have had on development teams. We list the features of a good publishing framework and define the additional requirements of a true content management system (CMS). We also list the most popular open-source publishing frameworks currently available and provide references for each.

The Challenges of Web Publishing

The TCP/IP network called the Internet had been in use since the late 1960s by an exclusive group of individuals affiliated with the government, academia, research institutions, and technology companies. The costs and complexity associated with using this network prevented the average person from understanding and using it as a tool for increasing personal productivity. In 1992, the adoption of the Hypertext Markup Language (HTML), derived from the more complex Standard Generalized Markup Language (SGML), caused a lot of excitement in the technology and business communities. The complexity of using the Internet was abstracted by the creation of a network of servers that could be used for publishing documents based on HTML and its parent, SGML. This network of information servers came to be known as the World Wide Web.

Any document could be marked up using a set of standardized tags and published to a web server. A remote client browser application could access this document, interpret the tags, and then display the information. The web developer predefined the format of the document and changing the presentation involved modifying the markup tags in the document. Vendors began providing their web server applications and browsers for free and this resulted in the proliferation of web technologies. The ease with which a developer could mark up a document and the availability of tools for the nontechnical content developer made HTML a very popular language for web publishing. HTML soon came to be the *lingua franca* of the Web.

Dynamic Presentation and Browser Wars

With the adoption and increasing popularity of HTML came the inevitable battle among vendors for control of the standard. In a move to protect their installed bases, vendors began to support nonstandard tags and server configurations. Developing content for consumption within an intranet was slightly less complicated, because most organizations exercised some control over the browser versions deployed within their enterprises. As time went by, the number of browser vendors increased, as did the versions of a particular browser. The nightmare had only just begun for content developers.

HTML was an excellent mechanism for authoring and rendering static information pages with graphical images, but it had some serious limitations. It did not address dynamic presentation or provide any way for the application to access data from external sources. Dynamic presentation and browser customization was made possible by the introduction of support in HTML for scripting languages like JavaScript and VBScript. Server-side technologies, such as the Common Gateway Interface (CGI), Java Servlet technologies, and a few others, enabled dynamic data to be displayed in browsers. Prior to the introduction of these technologies, the website was a tool for information distribution and was only used to support the marketing functions of an organization. Not long after dynamic content made its way into browsers, the industry presented strategies and products to help organizations make money on the Web. Up until that time, only the big companies could afford to conduct business electronically with expensive and proprietary technologies based on the Electronic Data Interchange (EDI) standard. The new web applications allowed even the smallest players to get into the electronic commerce arena.

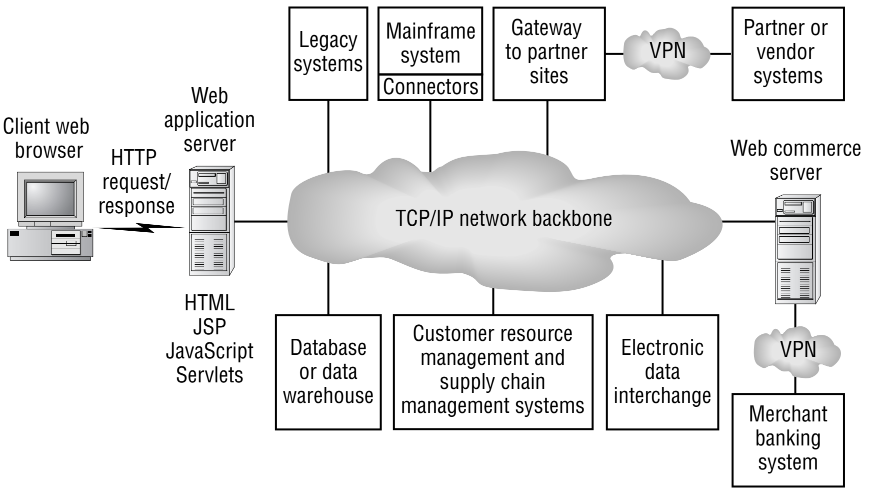
Enterprise Applications and Dynamic Data

Web application servers became more sophisticated and could now perform multiple tasks, such as serving up information pages customized for a particular browser and embedding data extracted from one or more data sources in those pages. The servers allowed users to initiate transactions and execute business processes—for example, making a retail purchase or placing a bid on an auction site. The user did not need to navigate to each business server, extract individual pieces of information, and manually correlate all relevant data. The service provider had to develop an application that collected all the data behind the scene and presented the user with an integrated and unified web page with the data requested.

The web browser paradigm abstracted the complexity of the organization's technical infrastructure by presenting the results of all operations in the form of a web page with some dynamic presentation data objects. The web server was promoted from the old role of a static electronic brochure to a dynamic catalog of the organization's products and services. It now provided online service delivery and managed user sessions and transactions. It also provided a front end for managing user relationships and communications with the organization.

Figure 1.1 shows a layout of a web application that interfaces with different enterprise services and data stores. It shows the web server as the repository of all the parts of the web application.

FIGURE 1.1: A typical enterprise web application



The Challenges of Web Content Management

Figure 1.1 shows the complexity and needs of a typical web application. This section looks at the challenges facing web application developers. It discusses the issues associated with integrating enterprise services as well as the problems associated with managing change, and highlights some of the ways development teams are dealing with these issues.

Integrating and Formatting Data from Multiple Sources

One critical issue that presented a challenge when developing web applications that interfaced with enterprise applications and legacy systems was that data in most enterprises was not available in a single, easy-to-use format. The data had to be extracted, modified, and reformatted before it was suitable for presentation in a web page. Java offered some relief by providing support for developing complex multitier applications. Technologies like Java Database Connectivity (JDBC), Servlets, and Java Server Pages (JSP) provide a means to seamlessly integrate disparate legacy applications and database systems while sticking closely to the web presentation paradigm—that is, output in the form of HTML pages.

This was a double-edged sword and highlighted another critical issue: In an HTML page, content and presentation are so closely tied together that it is difficult for external applications to extract the data embedded in both static and dynamically generated pages. Both these issues were partially addressed by the adoption of Extensible Markup Language (XML) as the standard for modeling data in enterprise and web applications. The use of XML allowed for separation of content and presentation. Because XML documents are normal text files, Java Servlets can be used to dynamically produce XML instead of HTML. Customized presentation was achieved through the use of either Cascading Style Sheets (CSS) or Extensible Stylesheet Language (XSL).

Change Management and Content Management

All these requirements added to the challenges of designing and implementing web applications. As the size and complexity of applications increased, the two issues that moved to the forefront were content management and change management. Content management deals with managing all the objects and parts that go into building the complete application. The use of Java and XML technologies enables the developer to produce robust and easily integrated building blocks for the application. Change management addresses the issues associated with modifying or enhancing the application after it is in production. Some organizations have very stringent requirements, which dictate that changes happen frequently and must be implemented very quickly. This not only includes changes to the dynamic data on the enterprise systems but also how the data gets presented on the pages.

Tighter integration between the web application pages and the enterprise applications or data systems allows for more dynamic data to be available in the final presentation, as in the case of inventory control and purchasing websites. Most content pages have dependencies elsewhere in the same application, or some other application on the same web server, or other applications on remote servers. There is a ripple effect when changes are made to these applications, and often a simple change affects multiple documents, servers, and departments within an organization. The complexity and size of an average dynamic website necessitates either a larger change management team or the automation of the process of change management.

Taming the Web Beast

Most organizations responded to the challenges by increasing the size of the content management team and settling on standardized technologies as the basis for their applications.

An implied challenge was choosing development tools that strictly adhered to the standards and training the entire development team in the use of these tools. Despite all the planning and strategies, content changes rarely could keep pace with the rapidly changing business needs. The limitations of the tools and the change management process often resulted in teams failing in their efforts to manage and maintain web applications. This resulted in a higher cost of ownership, decreased satisfaction on the part of the business sponsors, and an overall reduction in team morale.

Content Management Systems to the Rescue

The Web had proved its potential as a tool for delivery of information and services over the Internet. Organizations of all shapes and sizes quickly jumped on the bandwagon and adopted web strategies. With this came the big push to web-enable products and enterprise applications. Even vendors of proprietary solutions caught the wave and began offering applications designed exclusively for the Web, or at the very least, providing a web front end for their products. The large sites were rich in data and dynamic presentation and created new challenges for the teams designing and maintaining them.

The emphasis was now on teams that were composed of specialists in a particular part of the web application. One such team would specialize in the look and feel of the web pages and the related technologies like HTML and scripting languages. Another team would specialize in integrating enterprise applications into the web application. Newer technologies and standards were hastily put together to support the development and maintenance of large websites. However, this strategy came with its own set of problems. The size of the teams increased and the people-management aspects became a big issue. There were also problems of bridging the training and knowledge gap between teams.

Separation of Concerns

There were sections of the web application that included pieces that mixed the two functions. For example, Java Servlets provided the means to integrate with enterprise applications but would also be used to create the dynamic output to the client. This placed additional pressures on the application integrators because they now had to be concerned with graphical user interface issues. One of the design goals of Java Server Pages (JSP) was to separate content from presentation. However, this came up short and only added to the confusion of the development teams. JSP put additional pressures on the presentation developers because they now had to learn the syntax of Java and had to deal with programming issues. An interim solution was to create cross-functional teams in which members were trained in presentation and application programming, but this was a very expensive solution. There was a need for a solution that would allow the separation of the three areas: content, logic, and presentation.

Web-Publishing Frameworks and CMSs

A web search on the term *content management system* (CMS) reveals a long list of products and frameworks that advertise an easy way to manage the large quantities of information and the services that an enterprise has to offer to its customers. Several of the products and organizations use the term *content management systems* interchangeably with other terms like *content-publishing frameworks*, *web-publishing frameworks*, and *XML-publishing frameworks*.

This section provides a general definition of CMSs that encompasses all the design requirements of a good system. The definition highlights the features of the system that address and solve the challenges created by the surge in the popularity of the Web as a vehicle for delivering information and services. We also name a few of the products available and then shift our focus to the Cocoon architecture. We have chosen to focus on this one framework because it is currently one of the most powerful publishing systems and has the potential of becoming a CMS.

Defining CMS

A CMS is a framework that allows the creator or owner of an information delivery application to effectively manage all the pieces that go into building that application and that define formal processes that support the entire lifecycle of an application. CMSs and web-publishing frameworks offer solutions to most of the common business problems discussed earlier that are associated with developing and maintaining large web applications. They address the issues of rapid application development that offer flexibility in the look and feel as well as added functionality. Using a CMS as the central focal point for web application development strategy allows an organization to produce flexible and scalable applications in a cost effective and timely manner.

A CMS also allows seamless integration of tools and strategies that enable the creation of routine maintenance or upgrades, and the eventual retirement of the application. The CMS integrates data from multiple applications and services and provides a flexible mechanism to format the output and present it to the user. The technologies and strategies might be standards-based or proprietary. The systems that have the term *web* in the name support the standard web technologies and, at the very minimum, use HTML and the Hypertext Transfer Protocol (HTTP). In addition to supporting the Web, the frameworks can support alternative delivery mechanisms, such as dedicated client applications, whether they are stand-alone windows, fat clients, or Java applets embedded in web pages. They also support the integration of the output into another application framework such as Web Services. A few of the products satisfy only a subpart of the definition and might rely on proprietary technologies for the management and delivery of the information and services. Some of the frameworks include specific technologies in the product name itself, such as XML, to emphasize support of the standards for formatting and presenting documents and data. This focuses on technologies that are standards-based and products that are part of the open-source development paradigm.

Web Server Versus CMS

Some of the products attempt to classify regular web information servers as CMSs. This might not be a wrong classification, because creating and managing websites that serve up HTML and images is not trivial given the potential size to which the website can grow. Change management in websites is a significant issue because hyperlinks in HTML pages create interdependencies and any process that enables an easy maintenance strategy needs to get honorable mention. However, the term *content management* is more encompassing and addresses systems and processes for managing information in many different formats and access to enterprise services from multiple sources.

Choosing a framework that offers all the features of a CMS for a website with static HTML content might be overkill, especially if the content is not expected to change over time and the pages do not need to integrate data from external applications. The complexity of configuring and maintaining such HTML content sites using a CMS cannot be cost-justified in the long run. There are several commercial or open-source web servers that are more cost effective and suited for simple websites serving up static HTML pages and images with a little dynamic presentation using scripting languages like JavaScript or VBScript.

A Brief Review of Open-Source CMS Offerings

The following are brief descriptions of some of the open-source CMSs that are currently available and the web addresses of the organizations that support the development of each. It is not a comprehensive list, and their appearance is not meant to be an endorsement over any other open-source products.

XPS Extensible Publishing System from Wyona is an application that uses XML and Java technologies to manage documents and images on the server. Wyona supports XPS at `www.wyona.org`.

eZ Publish From eZ Systems, eZ Publish is advertised as an information management system with the data residing in a database. eZ Publish is created and maintained by eZ Systems at `http://developer.ez.no`.

Zope Developed by the Zope community, Zope enables teams to collaborate in the creation and management of dynamic web-based business applications. Zope is offered by the Zope community at `www.zope.org`.

Cofax Created by the Content Object Factory, Cofax is advertised as a web-based text and multimedia publication system. Cofax is available from the Content Object Factory at `www.cofax.org`.

Midgard From the Midgard Project, Midgard is defined as a toolkit for building dynamic applications for powering e-business and information management processes. Midgard is available at `www.midgard-project.org`.

MMBase MMBase offers a flexible solution for creating and maintaining big websites easily. MMBase can be found at `www.mmbase.org`.

OpenCms From the OpenCms Project, OpenCms is advertised as a Java-based web CMS and emphasizes the ease of creating and publishing web content. OpenCms can be found at `www.opencms.org`.

Why Choose Open-Source Projects?

As stated earlier, we emphasize the open-source frameworks that are helping shape the field of content management. Open-source systems are popular with most enterprises because of the lower costs of ownership. These systems are based on standardized technologies and are vendor independent. Most of the systems have a process for soliciting needs and requirements using a community process. The features invariably are based on needs that solve real problems and not on what a vendor decides is good for you. The developers of the frameworks are professionals in the field who have solved the content management challenges in their work environment. They bring a wealth of knowledge and experience from their careers to the development of these systems.

The Original Cocoon Project

The Cocoon Project started as an attempt to organize and control the documentation of all the projects being run under the Apache umbrella project. The first iteration was simple and was based on proposed technologies that had not yet been standardized. As the technologies evolved, the framework evolved along with it to utilize the new standards and include more developer requirements. The project recently released the second generation of the product, Cocoon 2, which offers more flexibility and features. We start with the first release, Cocoon 1, and examine the design goals, successes, and drawbacks.

Cocoon 1—a Simple Solution

By design, Cocoon 1 (C1) was based on the open technologies adopted by the Apache Software Foundation and utilized existing frameworks. C1 was a publishing framework that was written completely in Java. It was based on technologies standardized by the Worldwide Web Consortium (W3C), such as Document Object Model (DOM) parsing, XML for formatting data, XSLT for transforming data and merging XML documents, and Extensible Stylesheet Language (XSL) for presentation.

Cocoon was originally a very simple Java Servlet with approximately 100 lines of code and the format for the documents was XML. It used the IBM XML4J parser for parsing of XML documents and the LotusXSL parser for transforming the XML file using an XSL stylesheet. The next chapter goes into the details of the technologies that are the building blocks of the application, but we mention each of them in this section. The framework was defined with the adopted standard at that time. When the need arose for a server engine that would utilize XSL for transforming XML documents, the project was formally adopted by a vote on the jserv-dev mailing list and named the Cocoon Project under the Apache umbrella.

The Cocoon 1 Architecture: Strengths and Drawbacks

The Apache Avalon Project is part of the Jakarta Project and was an effort to allow developers of open-source projects to collaborate and share code easily. It created a common extensible framework and a set of pure Java components that could be extended to create new applications. C1 was based on the Apache Avalon framework and continued to have a simple structure with very little code. It was primarily used to demonstrate the importance of XSL and XML in web-based publishing. The Apache Xerces parser replaced the XML4J parser and the Apache Xalan parser replaced the LotusXSL parser. As the number of developers grew and additional requirements were added to the design goals, the simple servlet evolved into a complete XML-based publishing system. The framework was adopted widely and was used in production websites all over the world. The strengths of the framework were its simplicity and the fact that it was based on existing, popular frameworks.

However, the framework was based on XML technologies that were still in their infancy. The available parsers were based on the DOM and had several critical architectural issues. Performance was an issue because of the use of the DOM parser, which parses the XML document and creates a tree in memory. This also created greater demands on the server resources when multiple documents had to be served up to several concurrent users.

These drawbacks were not unique to the Cocoon Project and were based on the limitations of the technologies available at that time. Developers of other XML-based applications were experiencing similar problems. The W3C is the body that introduces, regulates, and adopts standards for the Web. To address the problems associated with XML, several new standards were proposed and adopted.

One of new standards addressed how XML documents would be parsed using inline Simple API for XML (SAX) events, which would eliminate the need for creating the object model in memory on the application server. Another change involved splitting up the XSL standard to address three different areas: XSL Transformations (XSLT), Formatting Objects (XSL:FO), and XPath for defining subparts of an XML document. Armed with these new standards, the Cocoon developers embarked on a two-year project to redesign the framework that solves the architectural problems of Cocoon 1 while adding new and interesting features that expanded the system's capabilities.

Architecture of the Cocoon 2 Framework

This section describes the Cocoon 2 framework and analyzes each of the features that the improved system has to offer. We place emphasis on the features and strengths that make it a good CMS for developing versatile web applications.

An XML-Publishing Framework

Cocoon 2 (C2) is a powerful XML-publishing framework based on XML and XSLT. As previously stated, the C1 architecture was based on DOM processing of XML documents. The C2 architecture is based on pipeline processing of SAX events, which results in better scalability and improved performance of web applications deployed in the framework. (More details on pipelined processing will be in the next section, and details of SAX and XSLT are covered in Chapter 3.)

A centralized configuration system makes the tasks of creation, deployment, and maintenance of applications a lot easier for the developer team. C2 has a powerful caching system that is based on a flexible design in which the components can be dynamically configured. This prevents the components and rules from being hard-coded in the C2 system itself. When a user request is received, the caching system checks to see if the requested Universal Resource Indicator (URI) is available in the cache and delivers the cached content instead of sending the request through a pipeline for processing. This feature also results in better performance when serving static content.

Flexible Content Aggregation

The C2 framework continues to be available as a Servlet but is designed to be an abstract engine not tied to a particular platform or application server. This Servlet can connect to any existing Servlet engine or most web application servers. C2 allows the developer to create custom protocol handlers, which enables the application to connect to external sources that can be accessed and retrieved with a standard URI. Cocoon can also call itself recursively using the `cocoon://<some URI>` protocol. The C2 Servlet does not need to be invoked by name to be used but can be mapped to a name that will be referenced directly in the URI being invoked on the server. Thus, if `some-page.xml` were the resource being requested from `yourserver`, the URI would be as follows:

```
http://yourserver/cocoon/some-page.xml
```

In addition to the Servlet interface, C2 offers a command-line interface for running a batch process to create static content, which can then be cached, in the caching mechanism previously described. Pregeneration of parts of a website that will be static for the life of the application results in marked improvement in the overall performance of the application. Static generation of content can be used to handle content like Scalable Vector Graphics

(SVG) rasterization or for applying stylesheets to XML documents. The features and functions of the C2 system make it very easy to manage static content. With C2, developers can easily aggregate content from many sources.

Pipelines and Components

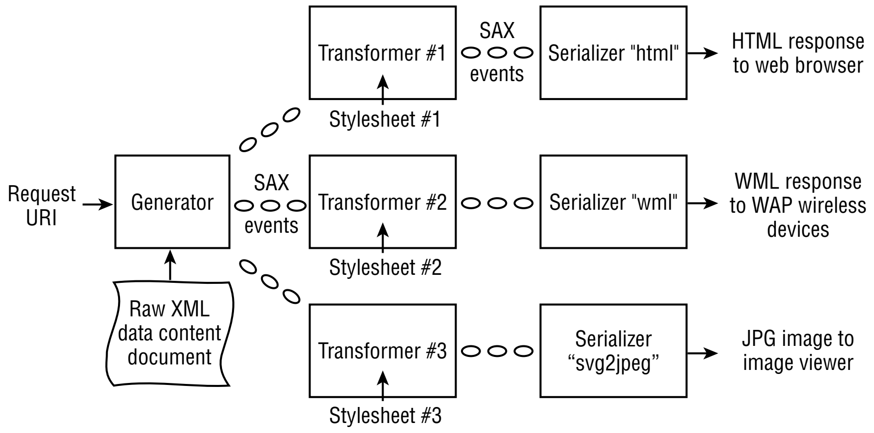
The C2 architecture is based on component pipelines. The concept of pipelines is similar to that used in the Unix operating systems. The difference is that in Unix, the elements in the pipelines were all bytes; in C2, the elements in the pipeline are all SAX events created by parsing XML documents. There are three types of pipeline elements, or *components*:

- Generators
- Transformers
- Serializers

Generators take a requested URI and produce SAX events. Transformers consume SAX events and produce other SAX events. Serializers consume SAX events and produce a response. Generally in C2, a pipeline is made up of one generator, zero or more transformers, and one serializer. At the very minimum, a Cocoon pipeline consists of a generator and a serializer Figure 1.2 shows a typical Cocoon pipeline with a generator followed by a transformer and a serializer for different types of output responses. It shows the SAX events that are produced and consumed by each component in the pipeline.

The generators, transformers, and serializers have been developed and donated to the C2 Project. These components are classes developed in Java, and all the basic ones come with the C2 distribution. The components can be extended to create new components, which inherit the basic behavior from the base component and add features of their own.

FIGURE 1.2:
A Cocoon pipeline



The following are the names of generators in C2. The names listed are the actual class names of the components in the framework.

`FileGenerator` is a parser, which reads a file or Uniform Resource Locator (URL) and produces SAX events.

`DirectoryGenerator` takes a directory listing as input and converts it to an XML format from which it produces SAX events.

`ServerPagesGenerator` generates XML and SAX events from XSP pages.

`JSPGenerator` takes a JSP page as input and generates the XML and SAX events.

`VelocityGenerator` takes the Velocity language templates as input and converts them to XML and generates SAX events.

The following are the names of the transformers in C2. The names listed are the actual names of the components in the framework.

`XSLTTransformer` takes a stream of SAX events as the input and transforms the events based on a given XSLT stylesheet.

`XIncludeTransformer` processes the `xinclude` namespace and includes external sources by adding their SAX events into the existing SAX stream.

`i18nTransformer` transforms SAX events in the pipeline using the i18n dictionary and a language parameter provided to it.

The following are the names of the serializers in C2. The names listed are the actual names of the components in the framework.

`XMLSerializer` takes the SAX events as input and generates an XML response as the output.

`HTMLSerializer` convert SAX streams in the pipeline into standard HTML output for web browsers.

`TextSerializer` produces text responses from text-based SAX events. This is useful for when the output does not have to be formatted and is mostly composed of non-XML text as in Cascading Style Sheets (CSS), Virtual Reality Modeling Language (VRML), and code text.

`PDFSerializer` creates an Adobe Portable Data Format (PDF) response stream using the Apache FOP (Formatting Output Processor). The Apache FOP uses the XSL:FO SAX events in the input stream.

`SVG2JPGSerializer` takes SVG SAX events in the input stream and, using Apache Batik, creates an output stream in JPEG format.

Introducing the Cocoon Sitemap

The Cocoon sitemap is the configuration document that defines the pipelines and components available for use by web applications. The sitemap lets the C2 system match the incoming URI to a particular pipeline that processes the request and creates the desired output responses. The pipelines in turn define the components that produce the static or dynamic output.

The sitemap is an XML file and complies with all XML syntax and notations. An example sitemap is shown in Listing 1.1. We have chosen a simple example to launch the discussion of sitemaps and lay the foundation for an understanding of detailed sitemaps. An actual sitemap for most web applications will have additional code sections, which are discussed in subsequent chapters of this book. For now we provide only the most important sections to help you understand the concepts. We then dissect the sitemap and explain each section in detail.

Listing 1.1 A Sample Sitemap

```
<?xml version="1.0"?>
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
  <map:components>
    <map:generators default="file"/>
    <map:transformers default="xslt"/>
    <map:serializers default="html"/>
  </map:components>
  <map:pipelines>
    <map:pipeline>
      <map:match pattern="hello.html">
        <map:generate src="hello-page.xml"/>
        <map:transform src="hello-htmlstylesheet.xsl"/>
        <map:serialize type="html"/>
      </map:match>
      <map:match pattern="hello.wml">
        <map:generate src="hello-page.xml"/>
        <map:transform src="hello-wmlstylesheet.xsl"/>
        <map:serialize type="wap"/>
      </map:match>
    </map:pipeline>
  </map:pipelines>
</map:sitemap>
```

All the elements in the sitemap must belong to the same XML namespace. The code fragment shown below defines the namespace `http://apache.org/cocoon/sitemap/1.0` for the sitemap and assigns it to the prefix `map`:

```
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
```

The sitemap defines the components and pipelines available to the application as child elements of the `sitemap` element. The following code fragment from Listing 1.1 highlights the `components` element and defines the components that are available for use in the Cocoon system.

```
<map:components>
  <map:generators default="file"/>
  <map:transformers default="xslt"/>
  <map:serializers default="html"/>
</map:components>
```

The code defines the default components that are available within Cocoon. The element `generators` has an attribute named `default` and denotes the default generator for the server. The value of the default attribute is `"file"` and designates the component `FileGenerator` as the default generator for the system. The next element is `transformers` and the `default="xslt"` attribute sets the default transformer to be the component `XSLTTransformer`. Finally, the `default="html"` attribute of the element `serializers` sets the `HTMLSerializer` as the default serializer.

A sitemap can define more than one pipeline for processing as shown in the following:

```
<map:pipelines>
  <map:pipeline>
    <!--pipeline #1 code goes here -->
  </map:pipeline>
  <map:pipeline>
    <!--pipeline #2 code goes here -->
  </map:pipeline>
      ...
  <!-- additional pipeline elements defined -->
      ...
</map:pipelines>
```

For simplicity, Listing 1.1 shows the `pipelines` element as having only one pipeline child element.

Matching the URI to a Pipeline

Within the pipeline there are one or more match blocks, which define the actual processing path for a given request. The following code fragment shows the pipeline with the match elements in our code listing.

```
    <map:pipeline>
      <map:match pattern="hello.html">
        <!-- match elements for processing the request -->
    </map:match>
    <map:match pattern="hello.wml">
      <!-- match elements for processing the request -->
    </map:match>
      ...
    <!-- other match blocks -->
      ...
  </map:pipeline>
```

Once again, for simplicity, the `pipeline` element is shown to contain only two match elements. The pattern attribute of the `match` element is a string name of the resource being requested by the client in the URI. C2 matches the incoming request URI to the specific match element in the sitemap. The first match pattern for the pipeline is shown in the following code fragment.

```
<map:match pattern="hello.html">
  <!-- match elements for processing the request -->
</map:match>
```

When the server receives a request from the client for the URI, `http://yoursever/coccon/hello.html`, Cocoon matches the requested resource in the URI (`hello.html`) with the pattern attribute of the `match` element and invokes this path in the pipeline.

If the server received a request of the type `http://yourserver/cocoon/hello.wml`, Cocoon will match the requested resource (`hello.wml`) to the pattern attribute of the `match` element and invoke the appropriate path as highlighted below.

```
<map:match pattern="hello.wml">
  <!-- match elements for processing the request -->
</map:match>
```

We have kept the example sitemap very simple and have shown only two paths for processing within the pipeline. In reality, an XML document might need to be consumed by many more types of applications. In that case, the `pipeline` element will have several more match blocks, one for each type of output response that will be generated. Within each match block there will be a generator, transformer, and serializer for processing the request and generating the response.

Processing the Requested URI

Once the incoming URI is matched to a pipeline, C2 begins processing the request by processing each element within the pipeline. The elements within the match block specify the generator, transformer, and serializer that make up a processing path within the pipeline and create the specific response to the input URI. The actual components used for each step of the processing were specified in the components element of the sitemap as discussed earlier. The following code fragment shows the match block for the resource matching the pattern `hello.html`.

```
<map:match pattern="hello.html">
  <map:generate src="hello-page.xml"/>
  <map:transform src="hello-htmlstylesheet.xsl"/>
  <map:serialize type="html"/>
</map:match>
```

The code fragment that follows shows the match block for the second match element in the pipeline, which services requests for the resource `hello.wml`.

```
<map:match pattern="hello.wml">
  <map:generate src="hello-page.xml"/>
  <map:transform src="hello-wmlstylesheet.xsl"/>
  <map:serialize type="wap"/>
</map:match>
```

The `generate` element has an attribute `src`, which specifies the source file that is the input of the generator component. It should be noted that for the given pipeline, the `generate` element is the same for both match elements, and the same XML document is the input for both paths in the pipeline. This is highlighted in the following code fragment.

```
<map:generate src="hello-page.xml"/>
```

This XML document contains all the raw data for a particular section of the web application. For large web applications, the raw content will be appropriately segmented into different XML documents, each pertaining to some subpart of the whole application. For the purposes of this discussion the input file `"hello-page.xml"` represents a particular piece of a web application.

Each match element uses the same transformer component to customize the presentation for the target client application. In each case, the transformer uses a different XSLT stylesheet specified in the attribute `src` to transform the SAX events. In our example sitemap, the first match element is using a stylesheet that formats the output for presentation in an HTML browser as shown in the following code fragment.

```
<map:transform src="hello-htmlstylesheet.xsl"/>
```

In the second pipeline, the transformer is using a stylesheet for creation of content for handheld wireless devices and is shown in the code fragment that follows.

```
<map:transform src="hello-wmlstylesheet.xsl"/>
```

The final stage of the pipeline is the serializer, which takes the SAX events from the transformer and creates the output response that will be streamed to the target client application. The serializer has an attribute named `type` that specifies the format of the output response created by the pipeline. In the case of the first pipeline, the attribute `type="html"` specifies that the output response will be an HTML document for display in a web browser and is shown in the following code fragment.

```
<map:serialize type="html">
```

The serializer in the second pipeline has an attribute `type="wap"` and creates the response for wireless devices as shown in the following code fragment.

```
<map:serialize type="wap"/>
```

This concludes the description of the sitemap and the basic features of Cocoon. In a real application, the sitemap contains many pipelines and each pipeline has multiple match blocks to handle the generation of responses in many different formats. We introduce advanced features and elements of the sitemap in the rest of the book as we develop applications for the framework. As stated earlier, C2 is a powerful framework for content aggregation, which is enabled and supported by the features of the C2 sitemap. The developer can assign different namespaces to the different content sources and aggregate the output content into one document.

Integrating Business Services

Cocoon supports XSP as the language for business logic. Cocoon has a built-in XSP processor that evaluates and compiles XSP code embedded in XML files. The syntax of XSP is very similar to XML, but it allows the developer to embed Java code within the pages. This concept is very similar to JSP, which had special tags to mix Java code with HTML. XSPs can be used to separate out business logic from an XML document or they can be used to aggregate content from other applications and services. Cocoon also supports the extended Structured Query Language (SQL) XML tags for use in XSP pages. This enables the Cocoon application to have direct access to data in a relational database, and the final output of the database query is formatted in XML. We develop sample applications in subsequent chapters to demonstrate the use of XSP for integration of SQL data and integration with other Java applications.

Separation of Presentation, Content, and Logic

The C2 architecture supports strong separation of presentation, content, and business logic. This allows web development teams to be demarcated along functional tasks and responsibilities. Content developers create raw XML files without concern for how the content will be used. These XML files can also be created dynamically by external applications or XSLT transformations of documents. Another team of developers can create XSP logic sheets that integrate business logic into the application. Cocoon can aggregate the data from a combination of raw XML files or the output of these logic sheets. Finally, a team focused on the presentation aspects of the application can create the stylesheets used for transformation of the data into the final output. Individual teams can then have separate processes for managing the creation of their deliverables.

The next chapter provides additional details of the sitemap and provides examples of how to use the sitemap, components, and pipelines. We also lay the foundations for building powerful web applications using all the features of Cocoon.

Summary

This chapter took a brief look at the history of the Web and how web applications have evolved from simple static HTML pages to very complex applications that integrate with enterprise systems. As the Web revolution spread, enterprise web applications that had a mix of dynamic data, content, and presentation became popular. We followed the evolution of technologies that enabled creation of very complex and sophisticated frameworks and the evolution of content publishing from simple static HTML pages to highly integrated web applications based on XML that can have a flexible look and feel.

To manage the design, implementation, and maintenance processes and resources, it became imperative that there be some level of separation between presentation, content, and logic elements of the applications. This enabled the demarcation of teams based on skills, roles, and responsibilities.

There are a plethora of systems available, each supporting a set of technologies and a methodology for the creation and management of web applications. These range from simple and inexpensive web servers that serve up HTML documents to very complex and expensive enterprise web application servers. We looked at some of the requirements and design goals of publishing frameworks as well as those of content management systems and highlighted examples of the open-source CMSs available.

The Cocoon 1 framework started out as a simple document-management system but evolved into an advanced web-publishing framework, albeit with some drawbacks. The Apache Project redesigned and released the enhanced version, Cocoon 2, which solved many of the performance and configuration problems of the original framework.

Cocoon 2 is a powerful XML-publishing framework based on XML and XSLT. The following are the salient features of the Cocoon 2 framework:

- A centralized configuration system
- A powerful caching system
- Support for flexible content aggregation
- Support for separation of presentation, content, and logic

The sitemap is the central configuration mechanism that utilizes pipeline processing of XML documents. It uses prefabricated components such as matchers, generators, transformers, and serializers to process SAX events dynamically. It is an extensible framework and supports creation of custom components that extend the basic functionality of the system.

The next chapter expands on the concepts covered in this chapter by designing web applications that demonstrate the features of the Cocoon 2 framework.