7.2 Apache Cocoon

An example of a Web publishing architecture based on XML technology

http://xml.apache.org/cocoon

- Apache: "Java publishing framework"
- runs as a servlet (Java class under the control of a Web server, to handle incoming HTTP request)
- "XML application server" in Bourret’s categorisation (discussed in Section 7.1)
- "can publish XML to the Web but generally cannot receive data from the Web in the form of XML documents"

Cocoon Philosophy

- Separation of content, processing logic and style
  - in different files
  - their creation/specification distinct activities
  - to enhance system modularity and extensibility
  - to support co-operative site maintenance by groups with specialised skills
  - to allow re-use of solutions to different tasks
- Content, logic and style merged using XSL transformations

Typical Use

- Automatic generation of HTML through processing (static or dynamically generated) XML files
  - client-dependent processing possible, serving, e.g.,
    - HTML to "typical" web browsers
    - WML to WAP-devices
    - XML to XML/XSL aware clients
  - more sophisticated formatting (FOP -> PDF) also possible

Simple Example

- Different versions of content for different clients:
  - <html version = "1.0"/>
  - <!-- Fig. 16.1 in Deitel et al. -->
  - <cocoon-process type = "xslt"/>
  - <html-stylesheet href = "welcome.xsl" type = "text/xsl"/>
  - <html-stylesheet href = "welcome-wml.xsl" type = "text/xsl" media = "wap"/>
  - <myMessage>
    - <message>Welcome to XML!</message>
  - </myMessage>

Simple Example (2)

- Cocoon is preconfigured to handle various User-Agents of the HTTP request
  - explorer, opera, lynx, java, wap, netscape
- Style sheet welcome.xsl specifies the default processing,
  and welcome-wml.xsl (See next) processing for WAP devices

Style sheet welcome-wml.xsl

- "Different versions of content for different clients:
  - <html version = "1.0"/>
  - <!-- Fig. 16.4 in Deitel et al. -->
  - <xsl:stylesheet version = "1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    - <xsl:template match = "myMessage">
      - <!-- Specify an additional Cocoon step to format the result as WML(): -->
      - <xsl:processing-instruction
        name = "cocoon-format">
        - type = "text/wml"
      - </xsl:processing-instruction>
      - ...
  - </xsl:stylesheet>
Style sheet welcome-wml.xsl (2)

```xml
<wml><card>
  <p><xsl:value-of select = "message"/></p>
  <p>This page has been transformed from XML into WML by Cocoon's XSLT processor. </p>
</card>
</wml>
</xsl:template>
</xsl:stylesheet>
```

Cocoon Internals

- Based on the reactor design pattern:

Components of Cocoon Architecture

- **Request**: client's HTTP request
- **Producer**: produces an XML document from the request for the process
  - dynamically, or by loading a file
- **Reactor**: chooses processors/formatters to work on the document, based on PIs
  ```xml
  <?cocoon-process type="xxx" ?> and
  <?cocoon-format type="xxx" ?>
  ```

Components of Cocoon Architecture (2)

- **Processor**: transforms given XML document into something else. Examples
  - the XSLT processor
  - the XSP processor
    - evaluates XSP pages and compiles them into producers
    - considered a Producer in Cocoon 2
- **Formatter**: transforms the document into a stream interpretable by the client

Components of Cocoon Architecture (3)

- **Response**: encapsulates the formatted document and its properties (length, MIME type, ...)
- **Loader**: loads the formatted result when it is executable code
  - The same process used for creating producer components!
  ```
  (compiled server pages, written with XSP; more later)
  ```

Optimisations

- Performance and memory usage a major concern
- Processing requirements of XML parsing, XSLT transformations and formatting are heavy
- Dynamic server pages **compiled** to executable Java classes
- A special cache system
  - both static and dynamically generated pages served from the cache (if they are up-to-date)
- Efficient enough?
  - 57 Cocoon-based Web sites mentioned (April 2001)
XSP (Extensible Server Pages)

- Cocoon's technology for building dynamic Web applications
- **XSP page**
  - an XML document containing tag-based directives that specify generation of dynamic content at request time
  - transformed into Cocoon producers
- Supports embedding procedural code into static document content, similarly to other server pages like JSP or ASP
  - See example next

**XSP: Embedding code into pages**

- Generate a greeting based on the time of day:
  ```java
  <xsp:logic>
     String timeOfDay = (new SimpleDateFormat("aa")).format(new Date());
     if (timeOfDay.equals("AM")) {
       <xsp:content>
         Morning
       </xsp:content>
     } else {
       <xsp:content>
         Afternoon
       </xsp:content>
     }
  </xsp:logic>
  ```
- OK, but we can also avoid intertwining code and content (see next)

**XSP: Library Tags**

- Could replace the preceding with
  ```java
  <util:time-of-day />
  ```
- **<util:time-of-day />** is a library-tag
  - Extensibility!
  - Supports division of labour:
    - programmers encapsulate behaviour in dynamic tags
    - content authors use application-oriented markup to write XML documents
    - presentation designers develop XSL style sheets

**Example: Full XSP page**

```xml
<?xml version="1.0"?>
<!-- Generate a producer out of this page:-->
<?cocoon-process type="xsp"?>
<!-- Apply style sheet to the result of producer: -->
<?cocoon-process type="xslt"?>
<?xml-stylesheet href="sample.xsl" type="text/xsl"?>
<xsp:page language="java" xmlns:xsp="http://www.apache.org/1999/XSP/Core">
  <xsp:logic>
    Date now = new Date();
  </xsp:logic>
  <p>The time is now <xsp:expr>now</xsp:expr></p>
</xsp:page>
```

**Example: Result**

- Upon Cocoon processing (formatted according to sample.xsl), the page could be:
  ```html
  <html>
  <head><title>Time of Day</title></head>
  <body>
  <h3 style="color: navy; text-align: center">Time of Day</h3>
  <p>It's now Thu Dec 23 20:11:18 PST 1999</p>
  </body>
  </html>
  ```

**Implementation of Tag Libraries**

- Libraries of dynamic tags are implemented as corresponding style sheets. For example,
  ```xml
  <util:time-of-day format="yy/MM/dd hh:mm:ss aa" />
  ```
  could be implemented by template
  ```xml
  <xsl:template match="util:time-of-day">
    <x:sp:exp:formDate(new Date(), "<xsl:value-of select="/format" /">"));
  </x:sp:exp:template>
  ```
Implementation of Tag Libraries (2)

- Built-in XSP directives implemented by replacing them (via an XSLT stylesheet) by corresponding source code
- Compare:
  - Separation of document structure and presentation by style sheets that give presentation semantics to tags
  - Separation of document structure and processing by style sheets that give procedural semantics to tags

-> "Logic sheets"!