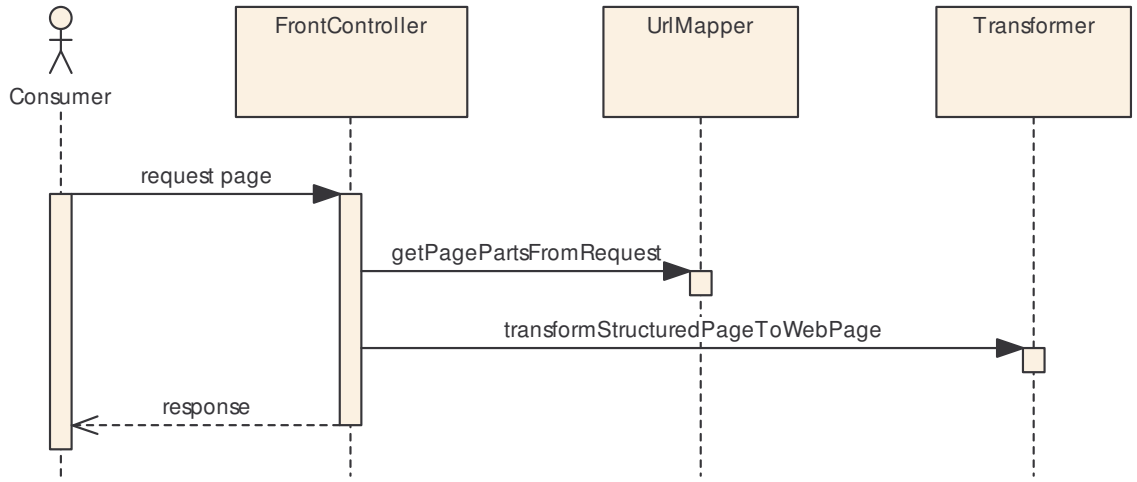
Stylized Page

Creates pages dynamically using pre-designed style templates. Maintains page design and layout fluid, allowing design to change without causing a huge impact on human resources.



Background

Web developers need to produce pages that have desired design characteristics. They need to add new content on a continual basis and have the new content reflect the same consistent presentation look and feel as well as elements as existing pages. If the team decides to use a traditional page design approach the will be a lot of work required for each page. A Web designer will have to layout each page independently of the others and create design elements specifically for that page.

Value and Benefits

The goal of this pattern is to reduce the amount of manual labor required to add new pages to a Web site. Whenever you can reduce the number of redundant steps needed to maintain a Web site that may contain hundreds or thousands of pages, the more your business will benefit. You will lower the cost and frustration associated with ongoing Web site maintenance.

There is the added benefit of providing a common look and feel to your site. You may insert banner and sidebars images with more ease and consistency. All articles or info pages on the site will have a common layout and speak well for the organization providing them.

The principles introduced here may also be applied to the *Business Document Publisher (page #)* pattern. This pattern does not address XSL-FO, but the same techniques can be applied to page layout, design, and generation of PDF and other printable document formats.

Putting It to Work

A request is received on your business *Dynamic Web Site (page #)*. The consumer wants to view a particular page that is described in the request. There are two basic strategies to this pattern, which I will now present.

Just-In-Time Page

An object that implements the *Front Controller* pattern [P of EAA] receives the request dispatch from the Web server. The *Front Controller* looks up the meaning of the URL. It may be part of the URL path or a parameter attached to it, but some part of the URL contains information about what the consumer is requesting. The `URLMapper` object maps the URL context to the information packet that describes the parts used to assemble the requested page. The *Front Controller* then uses the parts defined in the packet to create the requested page. An object of class `Transformer` is used to mold the request into a page response that fulfills the consumer's request.

Similar to the *Business Document Publisher (page #)* pattern, *Stylized Page* may use XML structured documents to serve content and XSL and XSLT to transform the XML into HTML pages. The `Transformer` reads XML and XSL, and passes the XML through XSLT to generate HTML. However, unlike the *Business Document Publisher (page #)* pattern you would not generate the structured document on the fly using this strategy. The XML and XSL will be created ahead of time and released to the Web site. Performance would be too poor to serve all content, back to front, completely dynamically.

And speaking of performance, one dilemma may need to be settled before you are satisfied with the implementation of this pattern. Should your *Dynamic Web Site (page #)* create *Stylized Page* output just in time, or should it publish pages out of band and serve them statically upon request? You will pay a stiff performance penalty if your Web site cannot produce pages just in time, but rather *barely in time* or even *never in time*. Hence, I next address the advantages of the *Pre-Generated Page* strategy.

Pre-Generated Page

The only real difference between this strategy and *Just-In-Time Page* is that with this approach pages are generated out of band from normal Web site operations. You simply have a production publish and release process that takes the XML input and passes it through the XSLT process to create static Web pages. The pages are then pushed out to the Web site using a release process. The *Front Controller* would then dispatch requests directly to the requested pages without performing a transformation at the time pages are served.

Personally I prefer this strategy to *Just-In-Time*. It may appear that it is not as easy to patch your Web site using this strategy. But is that really the case? Using either strategy you must have to perform publish and release. In the case of *Just-In-Time Page* the process will push out XML and XSL. In the case of *Pre-Generated Page* the process will push out static HTML or a *Dynamic Web Page (page #)* component such as JSP or ASP.

If you use this *Pre-Generated Page* strategy you will never have to change the process if the Web site ever increases load significantly.

Examples

The following is a very simple XML document with content that will be dynamically published in an HTML page (in this case, actually XHTML).

```
<?xml version="1.0"?>
<pattern>
  <name>Stylized Page</name>
  <summary>
    Creates pages dynamically using pre-designed style templates.
    Maintains page design and layout fluid, allowing design to
    change without causing a huge impact on human resources.
  </summary>
  <image>stylizedPageInto.gif</image>
</pattern>
```

There is purposely little content. There is just a name, a summary message, and an image. The important thing is to grasp what happens when the XML document is passed through a transformation engine in combination with a controlling XSL document.

Now note the XSL that is used to transform any XML document that conforms to the above structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                              xmlns="http://www.w3.org/1999/xhtml">

  <xsl:param name="contextPath"/>

  <xsl:template match="pattern">
    <html xml:lang="en" lang="en">
      <head>
        <title>Pattern Summary</title>
        <meta http-equiv="Content-Type" content="text/xhtml; charset=UTF-8"/>
        <link href="{$contextPath}/css/site.css" type="text/css" rel="stylesheet"/>
      </head>
      <body>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="name">
    <h1><xsl:apply-templates/></h1>
  </xsl:template>

  <xsl:template match="summary">
    <p class="block"><xsl:apply-templates/></p>
  </xsl:template>

  <xsl:template match="image">
    <p class="block">
      <img src="{$contextPath}/images/{@src}"/><xsl:apply-templates/>
    </p>
  </xsl:template>

</xsl:stylesheet>
```

As you may have put together, this example is a pattern summary viewer. It takes as input an XML document that contains a minimum amount of information about a pattern. In this example the pattern is the one you are reading, *Stylized Page*. The XML document has the pattern's name, the pattern's summary, and the introductory UML diagram image.

There is one main section to the XSL document. It is the one that matches on the outer `pattern` XML element. When the match is made then the basic HTML page is generated. Note that everything within the `<xsl:template match="pattern">`

element is text that is generated into the output stream, which is the skeleton HTML document.

Inside the HTML `body` element there is an XSL tag, `<xsl:apply-templates/>`. This means that whatever other content is matched in other template expressions, place them here in the order they are encountered. Now look at the final three template matches in the XSL document. One match is for the XML document's `name` element, one is for the `summary` element, and the last is for the `image` element. When those matches are made the corresponding HTML inside the XSL `template` element is generated, and the results are inserted into the main `body` region of the HTML document output stream.

Consequences

I again draw attention to these competing forces.

- ***Just-In-Time Versus Pre-Generated***: You will have to determine whether just-in-time page generation will have too heavy an impact on your specific *Dynamic Web Site (page #)*. This may depend much on the selected implementation. If the performance penalty is too high, you will need to use

Frameworks and Tools

- **Xerces**: This is an Apache open source project and is a very popular XML parser and document generation tool. To download see `http://www.apache.org/`.
- **Xalan**: This is an Apache open source project and is an XSL transformation engine. Xalan is well supported, and widely used even by commercial products. To download and learn more, see `http://www.apache.org/`.
- **Cacoon**: This is a Web page publishing framework provided by Apache. It is not limited to Web page publishing, and I therefore reference it in the *Business Document Publisher (page #)* pattern. The example in this pattern was adapted from examples provided in Cacoon 2.1.5.1.
- **.NET Tools:** [Provide list.]