Cocoon concepts

Table of contents

1 Comments.....5

Table of contents

1 Overview....3
2 Separation of Concerns (SoC)....3
3 Semantic markup....4
4 The sitemap....4
5 Flow....5
6 Cocoon Forms....5
7 Business logic....5

1. Overview

This document gives a brief overview of the most important concepts used in Cocoon.

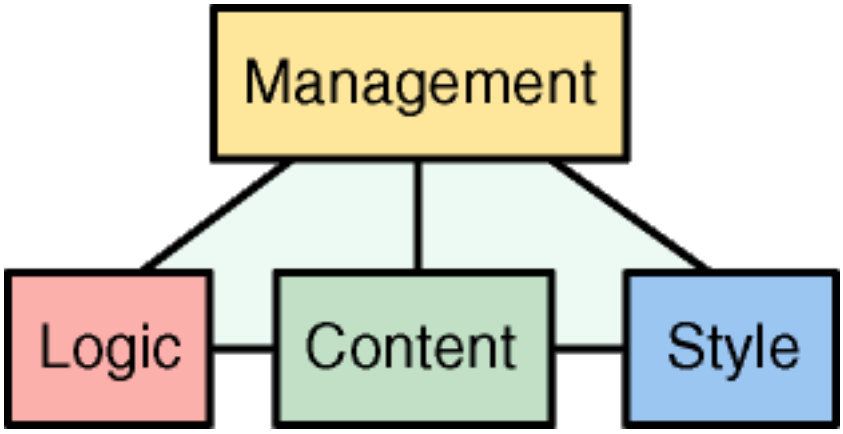
2. Separation of Concerns (SoC)

We believe the single most important Cocoon innovation is SoC-based design.

SoC is something that you've always been aware of: not everybody is equal, not everybody performs the same job with the same ability.

It can be observed that separating people with common skills in different working groups increases productivity and reduces management costs, but only if the groups do not overlap and have clear "contracts" that define their operability and their concerns.

For a web publishing system, the Cocoon project uses what we call the *pyramid of contracts*, with four major concern areas and five contracts between them:



The Cocoon Pyramid Model of Contracts

Cocoon is *engineered* to provide you a way to isolate these four concern areas using just those 5 contracts, removing the contract between style and logic that has been bugging web site development since the beginning of the Web.

Let's have an example:

<page> <content> <para>Today is <dynamic:today/></para> </content> </page>

Such a page is written by the content writers and you give them the "contract" that states that the tag <dynamic:today/> prints out the time of the day when included in the page. Content writers don't care (nor should) about what language has been used for that, nor they can mess up with the programming logic that generates the content since it's stored in another part of the system they don't have access to.

So <dynamic:today/> is the "logic - content" contract.

At the same time, the structure of the page is given as a contract to the graphic designers who have to come up with the transformation rules that transform this structure in a language that the browser can understand (HTML, for example).

So, the page structure is the "content - style" contract.

As long as these contracts don't change, the three areas can work in a completely parallel way without overwhelming the human resources used to manage them: costs decrease because time to market is reduced and maintenance costs is decreased because errors do not propagate out of the concern areas.

For example, you can tell your designers to come up with a "Xmas look" for your web site, without even telling the other people: just switch to the Xmas transformation rules on Xmas morning and you're done.... just imagine how painful it would be to do this on your web site today.

With the Cocoon architecture all this is a couple of line changes away.

3. Semantic markup

Although it is not a Cocoon invention, *semantic markup* is very important to work efficently with Cocoon.

By *semantic markup* we mean a way of building XML documents and models which preserves semantic information (metadata) as much as possible, by keeping the data in structured format and moving to presentation formats as late as possible in the document transformation process.

This document, for example: <page> <author id="3243">Will Coyote<author> <created>2005-03-12</created> <revision>1.42</revision> <content> Once upon a time, John made a bold move... </content> </page> contains structured information, that can be filtered and selected at Will to generate different presentations.

As you can see, there's nothing very sophisticated about semantic markup: the basic idea is to keep *everything that is known* about a given document or piece of information in a structured format. This makes it possible to precisely select the information elements that must be published, and to format them in as many ways as needed.

4. The sitemap

The *sitemap* is a central component of any Cocoon application, acting as a very powerful *request dispatcher*. It was the single most important innovation of Cocoon 2, and it is not going away soon: we *love* its power!

The main role of the sitemap is to trigger the execution of *pipelines* to process client requests. The sitemap uses *matchers* to select which pipeline to execute, matching various attributes of the incoming request (often the request path, but many other attributes can be used) to activate the first pipeline which matches the incoming request.

Here are a few examples of matchers: <map:match pattern="*.html"> ...pipeline definition goes here This pipeline would be activated for any request for a filename which ends in **.html*. <map:match pattern="**/resources/css/*.css"> ...pipeline definition goes here This pipeline would be activated for requests having paths like *something/resources/css/mystyles.css* or *a/b/c/d/resources/css/mystyles.css*

Sitemap variables give access to the matched parts of the request, for example to apply a common XSLT transform to all XML files found in a given directory: <map:match pattern="**/*.html"> <map:generate src="content/{1}/{2}.xml"/> <map:transform src="xslt/my-transform.xsl"/> <map:serialize type="html"/> </map:match> For the request *docs/planets/mars.html*, this pipeline would use the XML file *content/docs/planets/mars.xml* for input, process it with the *xslt/my-transform.xsl* XSLT transform, and send the result as an HTML document to the client. The double star (**) in the matcher pattern matches a path, while a single star matches a filename.

This last example also introduces the usual components of a pipeline: the *Generator* produces XML data, zero or more *Transformers* process the XML and at the end a *Serializer* converts the XML into the appropriate format and defines the *Content-Type* of the output.

Note:

The XML data passes as *SAX events* from one component to the next in the pipeline.

There's much more to the sitemap: it can contain component definitions, define *views* to make the various stages of the pipeline available for debugging, define *flowscripts* to act as the glue between pages, and several other things that we'll discuss at a later point. Sitemaps can also be "mounted" to create hierarchies of sitemaps and modularize applications.

5. Flow

FIXME (BD):

TODO: a brief description of this might be good here to have all the critical concepts in a single document.

6. Cocoon Forms

FIXME (BD):

TODO: a brief description of this might be good here to have all the critical concepts in a single document.

7. Business logic

FIXME (BD):

TODO: a brief description of this might be good here to have all the critical concepts in a single document.

Talk about the integration of Java code, REST backends, etc.

1. Comments

add your comments