Introducing Cocoon (2.1 legacy document)

Table of contents

Table of contents

Warning:
This document was copied as is from the Cocoon 2.1 documentation, but has not yet been fully reviewed or moved to its new home.

1. The XML Hype

Everybody talks about XML. XML here, XML there. All application servers support XML, everybody wants to do B2B using XML, web services using XML, even databases using XML.

Should you care about it? Given the amount of hype, you can't afford to go around ignoring XML, for that would be like ignoring the World Wide Web 10 years ago: a clear mistake. But why is this so for XML? What is this "magic" that XML seems to have in solving your problems? Isn't this another hype to change once again the IT infrastructure that you spent so much time implementing and fixing in the last few years? Isn't another way to spill money out of your pockets?

If you ever asked yourself one of the above questions, this paper is for you. You won't find singing-and-dancing marketing hype, you won't find boring and useless feature lists, you won't find the usual acronym bombing or those good looking vaporware schemas that connect your databases to your coffee machines via CORBA or stuff like that.

This document will explain you what the Cocoon project is about and what we are doing to solve the problems that we encountered in our web engineering experiences, but from an executive perspective, yes, because we all had the problems of managing a web site, dealing with our colleagues, rushing to the graphical guru to have the little GIF with the new title, or calling the web administrator at night because the database is returning errors without reasons.

It was frustrating to see the best and most clever information technology ever invented--the Web--ruined by the lack of engineering practices, tortured by those "let's-reinvent-the-wheel-once-again" craftsmen who were great at doing their jobs as individuals but could not scale within teams, imposing a growth saturation to their projects.

There had to be a better way of doing things.

2. Personal Experiences

In 1998, Stefano Mazzocchi volunteered to create the documentation infrastructure for the java.apache.org project, which is composed of a bunch of different codebases, maintained by a bunch of different people, with different skills, different geographical locations and different degree of will and time to dedicate to the documentation effort.

But pretty soon he realized that no matter how great and well designed the system was, HTML was a problem: it was *not* designed for those kinds of things. By looking at the main page (http://java.apache.org/) from the browser, you can clearly identify the areas of the screen: sidebar, topbar, news, status. But if you viewed the underlying HTML, boom: a nightmare of table tags and nesting and small little tricks to make the HTML appear the same on every browser.

So he looked around for alternative technologies, but *all* of them were trying to add more complexity at the GUI level (Microsoft Frontpage, Macromedia Dreamweaver, Adobe GoLive, etc...) hoping to "hide" the design problems of HTML under a thick layer of WYSIWYG looks.

What you see is what you get.

But what you see is all you've got.

How can you tell your web server to extract the information contained within the sidebar? How can you tell it to find the news articles within a complex HTML page?

It's certainly easy for a human reader: just look at the page and you should have no problem distinguishing between a sidebar, a banner, a news and a stock quote. Why is it so hard for a machine?

3. The HTML Model

HTML is a language that tells your browser how to "draw" things on its window. An image here, a letter there, a color down here. Nothing more. The browser doesn't have the "higher level" notion of "sidebar": it lacks the ability to perform "semantic analysis" of the HTML content.

Semantic analysis? Yeah, it's the kind of thing the human brain is simply great at doing, while computer programs simply fail at big time.

So, with HTML, we went a step up and created a highly visual and appealing web of HTML content, but we went two steps back by removing all the higher level semantic information from the content itself.

Ok, let's make an example... most of you have seen an HTML page... if not, here is an example:

```
<html> <body> <p>Hi, I'm an HTML page</p> <p align="center">Written by Stefano</p> </body> </html>
```

which says to the browser:

- I'm a HTML page
- I have a body
- I have a paragraph
- I contain the sentence "Hi, I'm an HTML page."
- I contain the sentence "Written by Stefano"

Suppose you are a Chinese guy that doesn't understand our alphabet, try to answer the following question:

Who wrote the page?

You can't perform semantic analysis, you are as blind as a web browser. The only thing you can do is draw it on the screen since this is what you were programmed to do. In other words, your semantic capacity is fixed to the drawing capabilities and a few other things (like linking), thus limited.

4. Semantic Markup

Suppose you receive this page:

```
<page> <author>sflkjoiuer</author> <content> <para>sofikdjflksj</para> </content> </page>
```

Can you now tell me who wrote the page? Easy, you say, "sflkjoiuer" did. Good, but later you receive:

```
<dlkj> <ruijfl>sofikdjflksj</ruijfl> <wijlkjf> <oamkfkj>sflkjoiuer</oamkfkj> </wijlkjf> </dlkj>
```

Now, who wrote the page? You could guess by comparing the structure, but how do you know the two structures reflect the same semantic information?

The above two pages are both XML documents.

Are they going to help you? Are they doing to simplify your work? Are they going to simplify your problems?

At this point, clearly not, rather the opposite.

So, you could be wondering, why did we spend so much effort to write an XML publishing framework? This document was written exactly to clear your doubts on this, so let's keep going.

5. The XML Language

XML is most of the times referred to as the "eXtensible Markup Language" specification. A fairly small yet complex specification that indicates how to write languages. It's a syntax. To tell you the truth, nothing fancy at all. So

<hello></hello>

is correct, while

<hello></hi>

is not, but

<hello><hi/></hello>

is correct. That's more than this, but I'll skip the technical details here.

XML is the ASCII for the new millenium, it's a step forward from ASCII or UNICODE (the international extension to ASCII that includes all characters from all modern languages). It defines a "lingua franca" for textual languages.

Ok, great, so now instead of having one uniform language with visual semantics (HTML) we have a babel of languages each with its own semantics. How this can possibly help you?

6. XML Transformations

This was the point where Stefano was more or less two years ago for java.apache.org: I could use XML and define my own semantics with <sidebar>, <news>, <status> and all that and I'm sure people would have found those XML documents much easier to write (since the XML syntax is very similar to the HTML one and very user friendly)... but I would have moved from "all browsers" to "no browser".

And having documentation that nobody can browse is totally useless.

The turning point was the creation of the XSL specification which included a way to "transform" an XML page into something else. (It's more complex than this, but, again, I'll skip the technical details).

So now you have:

XML page ---(transformation)--> HTML page ^ | transformation rules

that allows you to write your pages in XML, create your "graphics" as transformation rules and generate HTML pages on the fly directly from your web server.

Apache Cocoon 1.0 did exactly this.

7. The Model Evolves

If XML is a lingua franca, it means that XML software can work on almost anything without caring about what it is. So, if a cell phone requests the page, Cocoon just has to change transformation rules and send the WAP page to the phone. Or, if you want a nice PDF to printout your monthly report, you change the transformation rules and Cocoon creates the PDF for you, or the VRML, or the VoiceML, or your own proprietary B2B markup.

Anything without changing the basic architecture that is simply based on the simple "angle bracket" XML syntax.

8. Separation of Concerns (SoC)

Cocoon was not the first product to perform server side XML transformations, nor will be the last one (in a few years, these solutions will be the rule rather than the exception). So, what is the "plus" that the Cocoon project adds?

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* which outlines four major concern areas and five contracts between them. Here is the picture:

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.

Why? because programmers and graphic people have very different skills and work habits... so, instead of creating GUIs to hide the things that can be harmful (like graphic to programmers or logic to designers), Cocoon allows you to separate the things into different files, allowing you to "seal" your working groups into separate virtual rooms connected with the other rooms only by those "pipes" (the contracts), that you give them from the management area.

Let's have an example:

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

9. Here we go

If you've reached this far in my text, you should be able to grasp the value of the Cocoon Project as well as distinguish most of the marketing hype that surrounds XML and friends.

Just like you shouldn't care if somebody offers you software that is "ASCII compliant" or "ASCII based", you shouldn't care about "XML compliant" or "XML based": it doesn't mean anything.

Cocoon uses XML as a core piece of its framework, but improves the model to give you the tools you need and is designed to be flexible enough to follow your current needs as well as paradigm shifts that may happen in the future.

1. Comments

add your comments