

Making the Case for CFML

We've been on the right track all along

Many BlueDragon customers tell us they're being asked to defend their choice of CFML (ColdFusion Markup Language) over JavaServer Pages (JSP). They need help making the argument that CFML and J2EE work well together, and that perhaps CFML is a better choice for presentation layer technology than JSP for developing J2EE Web applications. They know instinctively it's the right choice, but aren't quite sure how to build the case.

As more organizations are standardizing on J2EE (and .NET) the issue of defending CFML will only become more urgent. The good news is that you've made the right choice with CFML. In this edition of **BluePrints**, we'll talk about why CFML can be a better choice than JSP for J2EE Web applications (in a future column, we'll take a look at why CFML can be the right choice of presentation layer technology for ASP.NET applications).

We'll see some of the reasons to stick with CFML, and identify some of the frequent arguments held against traditional CFML applications. We'll show how CFML has changed to address some of those arguments, as well as how design choices play a part in developing effective applications.

Then we'll look at the other side of the coin, discussing some of the common challenges of working with JSP – challenges that simply aren't an issue for CFML developers. We'll also see how JSP itself is changing, with the introduction of the Expression Language (EL) in the upcoming JSP 2.0 specification, and with the recently released JSP Standard Tag Library (JSTL). Indeed, the case can be made that JSP is looking more and more



By Vince Bonfanti

like CFML. Clearly we've been making the right choice all along.

This column will conclude with a brief discussion of why it doesn't even have to be a choice about "CFML versus JSP" at all. With both BlueDragon and CFMX, your CFML apps cannot only coexist with your JSP/servlet apps, but

the two platforms can benefit from tight integration with each other (BlueDragon also provides tight integration with the Microsoft .NET Framework, to be discussed in a future article).

CFML? What's CFML?

If you're a JSP developer who has happened upon this article, it may be useful to put CFML in context. Sometimes, people have gotten a misguided notion of what CFML is. Let's take a moment to recap. CFML really solves the same problems as JSP: it's a server-side scripting language for dynamic generation of Web content (typically HTML).

Created originally for ColdFusion Server (by Allaire, later acquired by Macromedia), CFML is no longer a single-vendor, proprietary language. New Atlanta's BlueDragon product line offers several implementations of CFML, which

allow you to run CFML on both J2EE and .NET servers, adding the possibilities of tight integration with native applications.

Anyone who's worked with CFML will agree that it's an incredibly productive language for Web application development. Designed from its inception to make dynamic database-driven applications easy for HTML coders, CFML has always been easy to understand. CFML code looks a lot like HTML. More important, CFML packs a lot of power into a few lines of code. Here's an example of a typical query and output process:

```
<CFQUERY NAME="Products" DATASOURCE="ProdDB">
    SELECT * FROM Products
</CFQUERY>

<H1> Product List</H1>

<CFOUTPUT QUERY="Products">
    #ProductName# $#Price# <BR>
</CFOUTPUT>
```

Of course, there are all the usual programming constructs (setting and using variables, control flow with ifs and looping, user defined functions, and more.) CFML is more than "just a language." It's also a framework with support for application and session variables, flexible database access and connection pooling, query caching, and more.

At the same time it's also a "high level" language incorporating features that on other platforms might require incorporating additional libraries or other products. Ben Forta's December 2002 *CFDJ* article, "But It's Free" (www.sys-con.com/coldfusion/article.cfm?id=541) discusses this aspect of CFML's richness in the context of comparing it with open source and other "free" platforms. As someone once said, CFML makes easy things easy and difficult things possible.

CFML's ease of use benefits not only new programmers but also coders responsible for maintaining someone else's application. Whether or not you're a CFML programmer, you can often readily tell what a CFML template is doing. It's almost self-documenting.

What's Not to Like?

While the tag-based nature makes it easier for nonprogrammers to pick up, some purists may argue that it's an ineffective mechanism for programming. Many developers don't find that it hampers their abilities at all. Still, the language has evolved to include an available CFSCRIPT tag to allow coding in a scripting approach based on ECMAScript (which is itself derived from JavaScript, though in the case of CFSCRIPT it's entirely server-side processing).

CFML's ease of use can be a double-edged sword though, as applications become more complex. As the earlier example shows, a query and its embedded SQL usually appear on the top of a page processing the query results. If the query page is the result of a form submission (such as a search page), it's typical to also throw the input validation onto the top of that same page. Some even go so far as to put the form and its processing on the same page, testing at runtime to know whether to show or process the form.

These methods are familiar to CFML developers, and they reflect the very growth of applications that start out simple and soon encompass more and more functionality. CFML makes it easy to prototype an approach, and there are times when its simplicity also makes it ideal for getting applications out the door quickly.

Still, the mix of business logic and display on a single page can have its drawbacks in larger and more sophisticated applications. And many don't even think about the problem because it's just always "how things were done" in CFML.

Indeed, to carry this scenario further, it's easy for any developer (especially a junior one) to create an application (especially a complex one) that doesn't perform well or won't scale. That's less often an indictment of the language than one of the developer's skills. (All those who've seen a JSP page run amuck doing too much and mixing presentation and logic,

raise your hands. We'll discuss in a moment the attempts in JSP 2.0 to solve that.)

Even so, many will also argue that there are times when segregating your applications into presentation and logic layers is overkill. Purist arguments make for lively e-mail list debates and academic discussions, but the reality for some organizations and for some applications is that the job needs to be done as quickly as possible. In such cases, CFML is just so much easier to get started with.

One final argument in favor of CFML, for a shop that already has a large base of CFML code, is that there's a lot to be said for the knowledge gained and the investment made in those applications. Migrating away from such apps will often be quite an expensive effort. We'll even see later that rather than convert the apps, you may want to explore integrating CFML with JSP applications.

JSP: The Challenges and the Changes

If you or your organization are set to proceed with JSP development anyway, or if you want some justification for holding off on it, let's take a look at some of JSP's challenges, as well as some recent changes in JSP development that influence this debate.

The first thing to consider, as a current CFML developer, is that getting into JSP isn't going to be a trivial process. Sure, JSP itself is a rather limited language and you don't really have to understand Java in order to edit/create JSP templates, but certainly you won't be able to do too much on a JSP page without knowing Java.

The primary goal of the JSP approach was to make it easy for HTML coders to edit presentation pages, with the thinking that the back-end processing (query processing, data validation, etc.) would be handled by a Java developer in a servlet or bean.

Unfortunately, many JSP developers are as unfamiliar as CFML developers with the notion of segregating page processing into layers. They too start to do more in the JSP page than just "presentation." The downside is that if they begin to embed Java code (the scripting language for JSP) onto the page, it will no longer be as easy to understand, edit, and debug.

Not only will a non-Java HTML coder have difficulty with the page, but even a novice Java developer and possibly an experienced developer will find it challenging.

This raises another point against JSP: its error reporting and handling leaves a lot to be desired. Because a JSP template is first parsed and then compiled, then finally run, there are lots of moving parts that could make it to track down the source of a problem. When you start to add more Java code to a template, it only exacerbates the problem.

Even if JSP developers start to follow the best practices recommendations of segregating an application into layers, they then have decisions such as how to encapsulate the "model," or indeed whether to go to EJBs. Then there's the question of whether and how to use servlets in addition to JSPs, such as for controllers, filters, etc. Add in the wide range of J2EE APIs available, and a "JSP" application can quickly grow into a very complex beast.

Most CFML developers are used to developing an application completely by themselves. So more than just having to learn Java, they have to become familiar with the process of segregating the application into layers. Again, CFML is evolving to support that notion, so it won't remain foreign for long. But in the near term, it means that it takes a lot more to understand all the implications and choices available.

For a discussion of some of these challenges, see the article, "Is Complexity Hurting Java?" by Jason Weiss in *CFDJ's* sister magazine, *Java Developer's Journal*. It's in the July 2002 issue at www.sys-con.com/java/article.cfm?id=1658.

Another take on the debate in that magazine, from the July 2001 issue, is "JSP: You Make the Decision," by Jon Stevens, at www.sys-con.com/java/article.cfm?id=1080. Part of this article is focused on introducing Turbine, one of many templating languages (including WebMacro, FreeMarker, and more) that have been proposed over the years as alternatives to JSP for Java Web application developers.

In supporting the value of these alternatives over traditional JSP, Stevens points out many of the challenges with JSP, reiterating the problem of embedding Java code within templates. He laments that

there's no way to prevent that occurrence in the JSP spec.

Stevens also points out that taglibs (JSP custom tag libraries) are an interesting solution, in that they provide a way to segregate such scripting into custom tags that are perhaps easier for non-Java developers to understand. But he laments that because developers are free to come up with their own, they may be reinventing the wheel. He recognizes efforts to create standardized taglibs, but at the time he wrote the article (nearly two years ago), such efforts were in their infancy.

JSP 2.0: Looking More Like CFML

A lot has changed since Stevens wrote that article. Indeed, the relatively new JSP 2.0 specification really changes the landscape of JSP programming quite a bit. We don't have room here to explain things but suffice it to say that the changes are pretty significant. The new expression language (EL) makes it so that referring to variables in JSP becomes a lot more like referring to variables in ColdFusion (though they're surrounded by `{..}` rather than `#` signs).

It also leans far more toward using less scripting, which addresses the concerns raised previously. It even offers a means to prevent inclusion of scripting in the page. Clearly, they're listening to the folks above complaining. You can learn more about JSP 2.0 at <http://java.sun.com/products/jsp>.

The JSP Standard Taglib (JSTL), introduces a new set of tags for setting and using variables, control flow with ifs and looping, and performing queries with embedded SQL. Hey! That sure sounds a lot like ColdFusion!

Indeed, an argument could be made that JSP 2.0 is an admission of failure of the original JSP spec and even that it's struggling to become more like CFML. Again, we've been on the right track all along.

Even with all these "improvements," there's still going to be more to creating a typical JSP/servlet application than just getting into JSP 2.0.

After hearing all this, are you sure you want to move to JSP after all? Is it really clear what it will buy you? If it's

becoming more like CFML, is it worth all the pain and expense of converting your applications to JSP?

Not an Either/Or Decision

When faced with the incursion by other technologies like JSP and .NET it's easy to fall into a trap of thinking that "defending" CFML is about choosing (or forestalling) a wholesale move to the other platform. It's not an either/or decision.

Whether yours is a JSP 1.x or 2.0 shop, or you're considering an alternative templating language, what if you could keep your current CFML applications and then consider whether to convert any aspects to JSP and/or the rest of the J2EE suite of APIs?

With both BlueDragon Server JX and CFMX Enterprise Server, you can run JSPs and servlets alongside your CFML templates and can even natively integrate the two. You can share session and application variables between CFML templates and JSP/servlets, you can transfer control and "include" information among them, and more.

And if you need to deploy your CFML applications on a J2EE server (such as WebSphere, WebLogic, Sun ONE, or others), there are J2EE editions of both BlueDragon and CFMX. BlueDragon's deployment approach offers advantages that will be important to most J2EE shops.

I've discussed all these points in my previous columns, – "It's Not ColdFusion – It's J2EE" and "CFML Forever," – both available at www.newatlanta.com/products/bluedragon/editorials.cfm.

CFML As a New J2EE Presentation Layer

Of course there will be those in the J2EE community who will argue that all the improvements in JSP 2.0 are just whitewash over a fundamentally challenged foundation. I agree.

What's needed is a mature, tag-based language; one that's equally suited to layered development or to simpler designs that may embed the model and the view at once. And that doesn't become more difficult to debug and maintain when you do that. Hmm. Where might they find one? Indeed, one


can absolutely make the argument that CFML can serve as a better presentation layer for J2EE applications.

Conclusion

If you have existing CFML assets, you can continue to leverage them effectively because CFML is a valid alternative as a native J2EE Web scripting language. You have many options.

You can stick with your CFML running on BlueDragon Server or CFMX and ignore JSP/servlets. If you've purchased BlueDragon Server JX (\$549) or CFMX Enterprise, you can keep your CFML applications and also run JSP/servlets alongside them, deciding at your own pace when to migrate or simply integrate those applications. And if you want to deploy your CFML applications on a J2EE server, you can do that as well and get all the benefits of such a platform while still preserving your investment in CFML.

In my next column, I'll introduce you to BlueDragon for .NET Servers. It's the only solution that provides all these same benefits of native integration in a .NET server environment. With our BlueDragon/J2EE and BlueDragon.NET products, we offer the only way to implement your CFML applications on both the major application server platforms.

Indeed, it may be the first Web application development language in the entire industry that's compatible across such a wide range of server platforms. That's the way to protect your investment in CFML. 

About the Author

Vince Bonfanti is president and co-founder of New Atlanta Communications, developers of Java- and CFML-based server products. A charter member of Sun's JavaT Servlet API and JavaServer PagesT Expert Groups, Vince has been a JavaOne speaker and a contributor to Java trade magazines and online publications. He has also been a featured speaker at Toronto's CFNorth and Washington's CFUN conferences as well as at local ColdFusion User Groups around the country.

vince@newatlanta.com