XmlcVersusVelocity

Case Study: XMLC vs. Velocity

A while ago, a question was asked on a Jakarta Tomcat mailing list about XMLC and Velocity. Bojan Smojver gave this thoughtful reply, and gave us permission to present it here on the Velocity site.

I found this in the tutorial for XMLC:

XMLC is a Java-based compiler that takes a document written in the Hypertext Markup Language (HTML) or the eXtensible Markup Language (XML) and creates Java classes that will faithfully recreate the document. The resulting Java classes can be used to insert dynamic content into the document framework at run time. XMLC, therefore, is a wonderful way to create dynamic HTML or XML documents from Java.

This sounds awfully like JSP to me, which is reason #1 why someone would use Velocity.

Although I use Velocity for my web work, Velocity is a generic template engine, it doesn't really care about the rest of the content. And that's great!

I use XSLT (sorry Jon, promise to give Anakia another look 🙂 to prepare my documents from XML into XHTML (this is not done at run time, not like Cocoon) and Velocity doesn't get upset much by that (except for the fact that you can't have '&&' (VTL and) in XSLT without applying a few tricks) but I've overcome that through Ant's nice replacement techniques...

Anyway, the first bad point to XMLC goes for the pains of code generation, which Velocity avoids so neatly.

The second is the actual process of designing the complete solution which is pictured here: http://xmlc.enhydra.org/project/aboutProject/index.html

Designer designs the page and then engineer puts in the logic? This sounds very bad to me. There should be a 'box full of Lego's' designers can choose their functionality from, not the other way around. If engineers have to be involved in simple projects, it comes back to employing engineers to do everything in the first place. And why wouldn't you use even JSP or ECS in such a case? You're are mucking around with Java again... The XMLC cycle is just too long and it doesn't make sense at all - what happens when a designer (by mistake or intentionally) screws up the id's in the page? All of the engineers code becomes unusable. Now that's a nice separation between presentation and functionality!

Second bad point.

An example from my 'production line', which illustrates how Velocity handles the above. I have a few classes that handle all inquiry forms:
one class handles the fields from the form and stores those into a database, the other one picks the fields and mails them to designated e-mail address. These classes are beans and are loaded from the only servlet I ever use (licensed GPL, but could contain any number of bugs since I wrote it), which handles ALL Velocity pages: 331 lines long including comments. Beans have scope and all, just like with JSP's.

Once the first ever form is created and debugged, an imaginary web designer on my 'production line' would copy and existing Velocity page from a previous project, change the fields (their names, their number... whatever) in it and the other content. Once they post the page on the site, that's it. It just works. They didn't even have to talk to an engineer to get things done. Now that's MUCH better then XMLC!

I think the biggest problem XMLC, JSP and servlets are facing is more of a philosophical nature: the document is data, not code - and Velocity treats it exactly like that. Why would you convert your HTML into a Java class when you want to send it to the browser as text?

My 2 cents. Velocity rocks!

Bojan