JimsRandomThoughts

The underlying requirement of all of the applications of dspace applications is for a rock solid repository of data and metadata. Ralph LeVan was espousing the idea of having the asset store responsible solely for storing objects and retrieving them by id (almost a la EJB). (Ralph - if I've misrepresented you here feel free to correct it!). I suspect it might be advantageous to add a few more areas of functionality than that, but it is important properties of the persistence API are that it's **stable** (i.e. needs a lot of thinking through) and represents a **lowest common denominator** of the persistence options available to us. That way that writing components to implement persistent storage to SRB, MPEG21 on the filesystem, RDF/XML with binary contents or even (unfashionably, I know) to an RDBMS is a) possible and b) a well defined task (and therefore implementable by a decent programmer in the future without having to understand the rest of the architecture).

Been looking at groovy; a scripting language that's easy and familiar for java developers. I've had some fun with jython before now, but got the feeling that the project was a little dormant (the lead developer has moved on, I hear). Groovy also has the fillip of a JSR. Basically it's a weakly (optionally) typed language based on java but with lashings of syntactic sugar, closures and some helpful extensions to key JDK classes that make things much more expressive. It also enables bits of functional programming in the same way python does (although it's closure support is much, much better). Groovy files compile to java, groovy classes can use java classes, and vice versa.

There are a few opportunities here for dspace 2.0 I think: -

- to implement the web application in something a bit more agile than Java (or at least to have the option). People are already starting to play around with groovy in cocoon, for example, and groovy comes with a Servlet that can automatically recompile and load your groovy classes when you edit them.

- Write key services in Java with hotspots for extension that could be easily done in groovy

- Define the component contracts and interactions in a Java framework (avalon?) and develop the components in whatever's most appropriate - they'll be deployed as jars in either case so it all comes out in the wash...

Been looking at the OpenSymphony stuff over the weekend (I do have a life, honest!). Looks good - I especially liked XWorks. At one level it's a component framework, and it would be interesting to find out the relative strengths and weaknesses between XWorks and Avalon. WebWorks looks quite smooth, and I'd consider it if I was choosing a web app framework to replace struts. Find of the weekend was SiteMesh though - the templating solution I've always wanted!

Spent part of today looking at JHOVE. I was thinking about doing a bit of work on a GDFR implementation myself, so it was timely to find out about this project. I wasn't going to be brave enough to write all those modules myself though - I was going to hack in support for that stuff by calling linux utilities. Looking at the size of the code required to do this stuff in Java, I reckon the idea still has legs; I can implement extra JHOVE modules for other formats in groovy (perfect for this kind of hackery).

Update - attempted this but a couple of bugs in groovy proved critical in thwarting my efforts. Have submitted bug reports and will wait for them to be resolved.

Having a look at the org.dspace.core.Email code and had a couple of ideas. 1) we should validate recipient email addresses as they're added. 2) How about named parameters for the templates? - easier to see what's going on in both template and code. Would mean we couldn't use MessageFormat, so probably not worth it unless we start needing more templates (or need a templating solution for other porpoises).

—

I'd really like to: -

- move the documentation into the same project space as the code, and shift the build system over to maven.

- remove all the import xxx.*; They just drive me batty.