The functionality I wanted for my interactive puzzle solving interface together with the requirement that it run in most browsers pretty much forced the choice of asynchronous JavaScript and XML (AJAX) or JSON . This allows a web page to communicae with the server and act on the response without doing a reload of the whole page.

(In many ways, this has become what java applets should have been, but java applets in the browser have been disabled by most users due to a string of security issues that have cropped up since they first came out.)

So, how to do it? I considered the following options:

- Hand-coded javascript, possibly using a javascript library like dojo . And using an java servlet which understands XML. However I'm not a javascript Guru, and, again, I don't partcularly want to become one (don't get me wrong; I can do it - I just don't like to...). Then again, if I do do it this way then I would have full control of what happens.

- Wavemaker application - This is an application builder that itself uses dojo, Spring and Hibernate. This allows a user to build database based applications fairly quickly and simply, however it was probably not flexible enough to allow me to do the sort of things that I want to do in this situation. In particular, creating widgets dynamically was not simple. If I had gone this way then I could have used various J2EE solutions for the interface to the back end, probably SOAP web services in this case.
- Google Web Toolkit (GWT) plus GWT-RPC (Remote Procedure Calls) plus a compatible servlet to do the back end processing. Not really open, but it does allow one to code in Java and the toolkit compiles the front end into javascript and looks after the implementation of the various different pats of the RPC mechanism. Probably the biggest drawback to this is that the RPC mechanism is not open - which might cause some problems when I come to implement the mobile phone versions of the client. Still that is a problem for another day.
- Same as the above (GWT or hand-coded JS), but using JSON - can't see why i would want to, but it is an option.
- Same client technology choices as above, but using Enterprise Java Beans . I didn't do this because the hibernate RPC method was simpler, but I could have and I may yet do so as an alternative implementation.

Also needing consideration was how to access the database (MySQL 5.x) assuming that the back-end code would be written in java. This came down to a showdown between JDO and Hibernate. They would both be working over JDBC and using JNDI. In the end, I chose Hibernate based on the prevalence of contracting job adverts which mention it. (I know - I'm shallow...)

In addition, I made the decision to use the joomla database to store the extra tables for the puzzle engine. This meant that the back-end also had access to the Joomla tables, particularly information about the registered users. I could have written the back end to access two independent databases, but - why bother?

Based on all the above, I came up with the following software stack (from the top looking downwards):

Joomla - for the website

Google Web toolkit - for the complicated pieces of user interface

GWT-Remote Procedure Call - for communication withthe puzzle engine

Java Servlet server - **Apache Tomcat** to run the puzzle engine.

Hibernate - to provide JAVA object persistence for the puzzle engine

JDBC + JNDI - to find and access the database

MySQL - the database

This made it possible to come up with a staged development sequence allowing me to get each layer working before adding the next. The sequence used was:

1. Stubbed Servlet - always returns the same hard-coded puzzle. Client displays the puzzle data.
2. Servlet accesses database. Tables are rpesent in the database and a static puzzle has been entered into the database manually.
3. Client provides interactive solving.
4. Submission of a solution. Servlet will return correct or incorrect based on an actual check against the real solution.
5. Add Puzzle submission interface

This is roughly the sequence I actually followed.

The next installment will look at setting up my development environment.