Joopal and Drumla

Sam Moffatt, Joomla!

September 13, 2009

1 Introduction

Joopal and Drumla grew out of a curiousity of mine. I wondered if it would be possible to integrate Drupal 6 into Joomla! 1.5 (hence creating Joopal, Joomla! is the primary system with Drupal underneath it) and once I didn't get much traction with that project after two major attempts, if I could put Joomla! into Drupal (Drumla under the same rules). This paper details the methods undertaken in the process, some of the interesting results and some learnings that were found along the way with the two projects.

2 Background

Drupal and Joomla! are two major web based PHP content management systems. As a Joomla! developer I hear all of this cool stuff about Drupal and the awesome extensions that they seem to have within their community. I do at times wonder why we don't have these extensions in our community but I'm sure that they are coming if we give them a bit of time. But then I had a brilliant idea: why don't we just encourage the Drupal developers to come along to the Joomla! side of the fence and start writing some really awesome hybrid Drupal/Joomla! applications. How hard can it be?

2.1 Joomla! structure

Joomla! as a CMS is comprised of various 'extensions' that can be installed to provide functionality in addition to the core supplied system. Joomla!'s extensions fall into a few categories: the main 'page' style extension is called a component. The component's handle the heavy lifting of the majority of the system and the user interface that is presented. If it is sitting in the middle of a Joomla! page, it is almost always a component doing something cool. Examples of components include the content component, the user component and the weblinks component. On the side of pages we have what we call 'modules'. Modules hang off the side of a page somewhere and provide utility style functionality. Examples of modules include stuff like menus, status indicators, login forms and footers. Modules on their own don't provide any major functionality, this is usually delegated to the components to handle. So the login form will send its data into the user component which also has its own login form and also calls the login logic that is available in the Joomla! framework. The last major code based extension is the plugin. Plugins are basically event handlers that respond to events and sign up to be notified of events. Examples of uses in the system of plugins include editors, authentication plugins and plugins which provide legacy library functions.

Components are called by simply including a file into the system - nothing particularly special. Modules also behave in a similar way though they have a different on disk location within the filesystem. Plugins behave differently by having a class that is initialised and then having the functions of that class called when particular events are triggered.

The way that plugins behave is actually how most people expect that hooks would behave in systems such as Drupal. In Drupal you might have a function called "modulename_block", a Joomla! plugin would provide this with a function called "block" within the class that the module was exposing. Plugins are also unique in the way that they can be scoped to different operations and only included as required however plugins can participate in a system scope and listen to requests from different parts of the system.

2.2 Drupal Structure

Drupal has the concept of a module which contains a set of code that should be executed in a single logical block. It uses the technique of having a few files within a module directory that are responsible for handling the way the module displays itself both as a page and as a block.

Drupal modules thus display two distinct extensions within Joomla!: the component and the module type. Within Drupal a Joomla! module is termed a 'block' and a component would be a Drupal module. So a Joomla! component is a Drupal module and a Joomla! module is a Drupal block.

As both the module and the block co-exist within the same file, Drupal has some logic which determines which 'hook' function to call and execute. For example to display a block in Drupal requires the creation of 'modulename_block' which returns a list of ways it can be displayed and also as an alternative option displays that information. Another hook registers menu callback functionality for links to the page display of the item.

3 Joopal

Joopal was the first project that I undertook. I had heard so much about how great Drupal was that I thought I would do something really cool: bring Drupal to Joomla!. In doing so I figured that I would get a deeper understanding of what made Drupal tick , how to take it apart and then how to put it back together again. At this point in time I think I have certainly succeeded in increasing my understanding of the underside of Drupal which has been a most depressing excursion of knowledge however it has taught me some new and interesting things.

3.1 Attempt 1

The first attempt at Joopal that I made involved me creating a new component directory within Joomla! and dumping a copy of Drupal in there with a slight change to add a single file that would cause Joomla! to load Drupal. To give the system a fighting chance I had taken it from an existing install and configuration to see if it would work. I had to make a slight alteration to the way that Joomla! behaved to ensure that the correct database was selected after the Drupal code was run and away I went. This method appeared to fail very heavily as Drupal's assumption that it was at the root of the system caused it to break almost all of the file includes. I tried for a day to try to get it to work and I didn't get much traction. I eventually gave up on that tact and decided to take a steadier approach to the matter. At this point I realised that integrating the systems together was going to be much harder than earlier expected as for Drupal's file inclusion to work within Joomla!, I'd need to at the very least rewrite how Drupal included its files.

3.2 Attempt 2

The second attempt to get Drupal to work under Joomla! turned out to be more successful. I wrote a small module for Joomla! that facilitated loading up Drupal blocks. It took me a lot of time and effort in hacking Drupal to get it to play nicely but eventually I got the basic Drupal logo display block working fine within a Joomla! module. So I thought I would try something more significant, so I started slowly adding more classes to the system to support more and more features. As I was going I started to rewrite parts of Drupal to be more object orientated with class versions of common functions and the ability to remove the plague of global variables that seem to infect Drupal so deeply.

I ran into problem after problem and my original goal of trying to get it up and running without touching significant portions of the code base were slowly dashed. The nail in the coffin was the fact that all over Drupal were includes that made the assumption that Drupal was being called from its own index.php file and that the file structure would be where it expected to be all of the time. I had discovered this in the first attempt but I didn't realise just how extensive this was. In this second attempt I had copied all of the modules across into a different directory under the Joomla! one to keep things clean. This so horribly broke parts of the system that it wasn't worth continuing. As a final fall of the hammer I gave up when I started to realise that Drupal was storing file system paths within its database. Some of the modifications I had made fixed up the problems but when I started having to rewrite entire chunks to ensure that things sort of worked fine I had pased my line. At this point it was becoming more and more like third party code wouldn't be able to run on it without very heavy modifications that I was trying to avoid.

The solution to this was to replace each of the PHP functions that load external files (include, include_once, require, require_once) with equivalents that automatically worked out how to handle the provided path. The equivalent functions check to see if the first character of the passed path is a slash or not. A slash would mean that the path is absolute where as the absence of a slash indicates that the path is probably relative. This presently doesn't work properly for Windows, however will work well on UNIX like systems (e.g. UNIX, Linux, Mac OS X). The functions pass absolute paths directly to the PHP equivalent built in function and then returns the result. Relative paths are prefixed with the Drupal root directory and then this is given to the PHP equivalent and the result is returned. This means that absolute paths work properly when being included and relative paths are also dereferenced back to the right place. The next problem with this approach is that the functions being replaced are built ins which don't require parentheses for their arguments so further modifications were required to handle these situations. This means that when a path is retrieved from the Drupal database it can either be relative (which is the default Drupal style) or absolute (the way that Joomla!'s influence is presently working).

So far this attempt is still in progress which is currently slowly porting Drupal into an OO interface which will fit better into Joomla!'s model better and provide the ability to easily handle items within Joomla!'s infrastructure. The benefit of this model in the long run is that it is then easy for Joomla! extensions to utilise Drupal functionality. Where possible ported functionality is using the newer style and so features like Drupal's Watchdog system is now presently independent and will eventually support Joomla!'s plugin infrastructure to demonstrate how the two systems are similar and can interact.

4 Drumla

So after giving up on Joopal for a while I decided that I would try my hand at making Drumla. Drumla took me about three hours to go from nothing to a Drupal module that loaded blocks from Joomla!'s module system and the ability for it to run Joomla!'s components. Those three hours weren't perfect mind you, there were still issues with the session system not deserialising objects properly after I had nuked Joomla!'s session system to prevent it from destroying Drupal's session object but this was solved by altering Drupal's bootstrap.inc file to include Joomla!'s object autoloader. Interestingly to get to this point I had hand crafted a Joomla! configuration file, loaded its SQL file into the system (I realise Drupal have a method for automatically creating the schema's however I didn't evaluate this point) and had extracted the base package out and added a Drupal module file and its support info file as well as two extra files, one for neatness and another to override the JSession class. The only change made to Joomla! itself was to edit the import loader file to prevent it from cleaning the request, this was effectively commenting out a single line, so that Drupal didn't have its information wiped out.

5 Conclusions

When considering application design we have two completely different situations. We have that presented to us of Joomla! which made integration into a third party application incredibly easy against that of Drupal which started to lead to a complete rewrite of the system to work around the inflexibility built deep into the system.

Overall, the integration of Drupal into Joomla! was complicated by Drupal's heavy use of relative pathing and storing copies of paths in the database. This effectively means that systems that discover these paths and include different files require significant updates to work properly. Another area of integration pain is in session handling. Both systems offer their own method of handling sessions so one needs to be disabled otherwise the two clash and cause issues. Particularly for Joomla!, part of its process involves cleaning out the entire session system which when being run within Drupal (Joomla! into Drupal) causes Drupal's information to be deleted.

Fortunately both systems feature the ability to respond to user events (e.g. creation/registration, editing, delete) which makes user synchronisation between the systems very easy.

So when designing a new system the following recommendations come to mind:

- Do not rely on a particular filesystem path existing, where possible implement a function or configuration item that enables a path to be specified to aide portability. This includes a relative path as well because an application might be running from a different location.

- Provide the ability to easily disable any functionality that could interfere with other systems (such as cleaning session or request variables). This includes the ability to defer session management to a different system properly.

- Provide the ability for different applications to adequately hook in and provide functionality. Particularly for users and sessions which interconnection between parts of the application.

Implementing these three features in an application provides the ability for another application to easily integrate. When looking at both Joomla! and Drupal, some of these features are missing. Drupal lacks the ability to easily be ported in the file system but easily provides the ability to replicate the session. Joomla! doesn't have an easy way of disabling its cleaning processes which interfere with Drupal.

6 Appendix

6.1 Drumla Changes

- Custom Joomla! Configuration:
  The live_site variable in the configuration needs to be set to the path to the Drupal site and then have "joomla" appended, for example:

  ```
  var $live_site = 'http://localhost/~pasamio/drupal/drupal-6.6/joomla/';
  ```

- Change to Joomla!:

  ```
  silversaviour:joomla-1.5.11 pasamio$ diff ./libraries/joomla/import.php
  33c33
  < JRequest::clean();
  ---
  > //JRequest::clean();
  ```

- Change to Drupal:

  ```
  silversaviour:drupal-6.6 pasamio$ diff ./includes/bootstrap.inc
  993a994
  >       include_once('sites/all/modules/joomla/bootstrap.php');
  ```