Topic Proposal

To: Ernie Doiron

From: Ryan Palmer

Subject: Work Term Report Topic

Date: September 7, 2007

Mr Doiron:

Purpose

This report is to be written as a case study for the implementation of a enterprise-grade Content Management System (CMS) built on the Drupal platform. Developing such a system has been my primary task while working for UPEI part-time since early January (full-time since July).

Significance

UPEI has chosen to publish its new public-facing websites using Drupal, and I have been instrumental in achieving this by developing new methods to help use Drupal in an enterprise environment. In this project we have overcome significant scalability and usability issues that we believe are of interest to the broader Drupal user community.

Audience

The primary audience for this report will be the broader user community of Drupal, not University stakeholders themselves. This report is targeted at an audience seeking specific technical information, so its readability will depend largely on the technical knowledge of the reader in the areas of

Linux, MySQL, PHP, Apache, and Drupal itself.

Research Plan

I have been developing in a Drupal-based environment for some time, and attempted to combine those skills with the existing capacities available through the UPEI Computer Science Department and the Integrated Promotions office. For this reason, there will not be a significant portion of this report based on external research, but of first-hand experience gained during the development stage of this project.

Ryan Palmer

Letter of Transmittal

To: Ernie Doiron

From: Ryan Palmer

Subject: Work Term Report

Date: September 7, 2007

Mr Doiron:

Attached you will find my Work Term Report not regarding my actual work term with EastLink Television, but regarding my employment with the University of Prince Edward Island, per our earlier discussions. If you have any questions regarding this report, please do not hesitate to contact me directly.

Sincerely,

Ryan Palmer

Drupal Implementation as an Enterprise-Grade Content Management System in an Academic Environment

By Ryan Palmer, Lead Developer

University of Prince Edward Island Content Management System Project

September 7, 2007

Abstract

The University of Prince Edward Island implemented an enterprise-grade content management system using Drupal in 2007. The objectives of the implementation were to provide a system that would be highly scalable, as well as maintainable by content managers of a beginner skill level, and themeable from a top level as to preserve the UPEI branding scheme on all sites.

Each site on the system was built as an independent site on its own database with no shared tables between instances, but run from a shared codebase. The sites were centrally controlled, and thus made scalable, by using LDAP Integration to authenticate with the existing authentication infrastructure at UPEI. Duplicate code in the codebase was also reduced by decreasing the amount of duplication in themes by using various filesystem techniques such as creating filesystem links to files versus copying files.

It was found that having an advanced understanding of the server environment that Drupal runs on and of Drupal itself were critical to the efficient scaling of Drupal sites, and that no module, core or contributed, was enough to ease the scalability issues that had to be overcome.

Introduction

Officials within the Computer Services Department and Integrated Promotions Department at the University of Prince Edward Island began looking at a campus-wide Content Management System in the Fall of 2006. Two Department and Faculty websites had been made using various content management systems, but neither were built with scalability in mind.

An internal estimate suggests that there are approximately 20,000 static web pages existing at upei.ca, spread over 300 website identities. These sites were either built by hand, or through the use of a WYSIWYG program. The challenges of maintaining this sheer volume of pages and sites were two-fold:

- to update content, either the content updater had to be skilled in HTML or possess the original WYSIWYG template files, resulting in a severe bottleneck for site updates, and,
- campus-wide updates of the UPEI branding scheme were not possible, nor was enforcing the UPEI branding scheme possible.

UPEI sought to overcome these issues by implementing a campus-wide content management system (CMS) that would incorporate user and permissions management and flexible theming.

Objectives

UPEI's objectives were typical of an institution centrally managed, yet departmentally autonomous: they wanted centralized control of the UPEI brand, but a permissions system flexible enough to allow various levels of

contribution. There was also a desire to keep each site within the system as an autonomous unit, so that functionality could be added, removed, or modified without affecting other sites on the system. This desired had to be weighed against the ease of which new features could be rolled out, system-wide.

The system also had to provide a centralized control of user authentication, so that authorized users could be granted access on the CMS, yet have authorization be revoked centrally if the user was to, for example, cease to be employed with UPEI.

There was also a desire to maintain a common look and feel between Departments. This would incorporate the UPEI brand elements (the crest and "swoosh"), as well as the navigation structure and page layout.

The CMS had to be robust enough to serve 30 million hits per month from a single server. In addition to the regular traffic incurred, the CMS had to be robust enough to serve the UPEI website in times of crisis, when tens of thousands of hits per hour would be anticipated. For this reason, it was elected to shield the CMS from seeing actual client traffic by using a static website copier to create static snapshots of CMS websites for delivery to outside clients. This decision was not made with Drupal in mind, but knowing that any websites served statically would be faster than pages generated from information in a database, as would be the case with any CMS.

The last major objective was to make as much content as possible "RSS enabled" so that it could be efficiently distributed. The very nature of

Department-based content was that it would be posted in irregular intervals. Most content would be static (program information, faculty information), but "Notices" posted at irregular intervals would be the content that would attract visitors back to the individual sites due to its timeliness and relevance. Because of its irregularity, RSS feeds were desirable so that others could "pull" content in directly whenever it became available. The topic of RSS is beyond the scope of this report, but suffice to say, Drupal enabled it effortlessly.

Drupal was chosen early in this project because of its ability to fill these needs efficiently. Its ease of implementation was what would make scalability issues easier to overcome.

Internal Resources

UPEI began this Drupal project with a set of institutional capacities typical for an educational institution. There was an extensive background in building static websites using various traditional and modern methods (NetObjects Fusion WYSIWYG editor), and an extensive graphic design resource at their disposal. On the technical side, there was an extensive background in Linux and Apache that would prove particularly valuable as strategies were developed to address the scalability issues that would challenge the project.

Implementation Team

- Glenda Clements | Manager, Integrated Promotions
- Blair Vessey | Operations Manager, Computer Services

- Debra Hannams | Web Designer, Integrated Promotions
- Virginia MacSwain | Web Designer, Integrated Promotions
- Ryan Palmer | Web Developer, Drupal programming lead, Integrated Promotions
- Neal Gillis | Co-op Student, Integrated Promotions
- Shawn Arsenault | Student Assistant, Integrated Promotions

Criteria for choosing Drupal

Compared to other Open Source CMS projects out there, Drupal seemed to have the right combination of a flourishing developer community on campus and around the world. It was chosen for its flexibility, and its ability (and desire) to interact well with 3rd party applications.

Drupal was also a known entity on campus:

- Business, Education and Applied Technology (BEAT) Program had implemented a campus-wide weblog built on Drupal in September of 2004. This site has grown to 1400 users, 5000 nodes, 8500 legitimate comments spread over 10 multi-site identities.
- Virtual Research Environment: The VRE has been developing Drupal sites as collaboration aids for researchers on and off campus for a few years. They had been active in developing themes and modules for some time. In addition, users familiar with the VRE would become easily familiar with UPEI CMS as their interfaces shared many similarities.

Implementation

The groundwork for building the UPEI CMS began in earnest in February 2007. At that point, the overall objectives of the project were identified and time was spent experimenting with various prototype configurations:

- Sharing Tables | The initial approach taken was with one codebase and one database for the entire system to run on. The majority of the effort then was spent slicing up parts of the systeem so that each multisite would appear as its own site, and to insure the owner of one site couldn't accidentally (or maliciously) post content to a separate site. Some of the things that were attempted were:
  - taxonomy access module in concert with a custom piece designed to automatically tag nodes based on the multisite they were created from
  - taxonomy redirect module to override term links with links to the multisite
  - inserted logic into node module to redirect a user's browser if a request for a node was coming from the wrong multisite

In June it was decided that a different approach would be taken, one that would see one Drupal codebase for all multisites, but a unique database for each multisite. This was based on recommendations from the University of Calgary who had went through their own Drupal roll-out process in early 2007.

Drupal Server Environment

A basic "LAMP stack" configuration was chosen for the server which Drupal would run on:

- Debian 4 "etch"
- Apache 1.3
- MySQL 5.0
- PHP 5.2

The Drupal Server Environment acted as a production and development server for all Drupal sites on the UPEI CMS. This server was generally not accessible from outside the UPEI campus firewall, although steps were taken to allow access for certain users.

Proxy Server Environment

- Debian 4 "etch"
- Apache 1.3
- PHP 5.2

The Proxy Server Environment acted as the front-end for www.upei.ca. All web requests for content at www.upei.ca would filter through this machine, and delivered based on a series of rules within the Apache configuration file. Content would be sourced either by proxying a legacy web server serving static html websites, or static snapshots of Drupal sites served from local folders on the machine. A nightly process running on the Drupal Server took static snapshots of Drupal websites, and used rsync to deliver them to the Proxy Server. This update process could be triggered by content

managers of each multisite as well.

Authentication

It was determined in the early stages of this project that authentication would be granted to all multisites by joining the Drupal sites with UPEI's existing Lightweight Directory Access Protocol (LDAP) server. This eliminated the need for user registrations, and allowed the roll-back of authentication of specific users to all websites centrally. The LDAP integration was accomplished easily using the LDAP Integration module and some pre-existing knowledge of the UPEI LDAP system.

Installation

The latest version (at the time) of Drupal grouped with the following key contributed modules:

- CCK content types
- Views
- LDAP Integration
- TinyMCE rich text editor
- Webform

As the upgrade process typically gets increasingly complicated with each additional contributed module utilized, each module chosen was evaluated based on not only its perceived usefulness to the typical multisite, but their predicted longevity within the Drupal community.

While the option existed, and was tempting, it was chosen that the use shared tables between multisites would not be used to assist in overcoming

scaling issues. The principal benefit in this case would be to centralize users by using a shared user table, but by using LDAP Integration instead most of the benefits were ruled out.

Another tempting use of shared tables would be to centralized taxonomy between multisites. It was determined that there would be no convincing need for taxonomy module at all, as the volume of content per multisite was not enough to warrant using it.

Multisites were installed by two methods: initially, it was elected to use Drupal Installation Profiles to install each site. This was desirable because it was part of Drupal core functionality. It was found to be difficult, though, to insure that all multisites ended up as intended, as some portions in the installation profile worked better than others. (Example: TinyMCE settings were difficult to implement via the installation profiles method) Approximately the first nine multisites were installed using this method.

The second method used was to develop various flavors of prototype (Academic Department, Administrative Department, Athletics Team) sites and to clone them to create new sites. Mysqldump was used to export the database structure and content, and a search and replace was done to change table prefixes. Finally, new MySQL databases were created and the sql files were imported to create the new website.

This decentralized approach of rolling out new multisites had serious drawbacks:

- it was difficult to implement changes to existing sites efficiently,

and,

- it was difficult to implement additional content types, pages and blocks in existing sites efficiently.

These issues were eased through the use of a custom module created to implement various portions of the functionality that was desired. Thankfully, Drupal is built in such a way that allows for functionality to be "dropped in" through the enabling of additional modules. This modules are able to interact with Drupal core, and shape the behaviour and output of Drupal sites. In the custom module, custom blocks, pages and permissions were defined that otherwise would have to have been distributed manually.

Filesystem

The Private Files system was chosen in the beginning as an effective means to restrict file access to certain files. There were no immediate plans to use this feature, so when it was determined that using this method slowed down the site significantly, the switch was made back to Public Files. This switch happened after approximately 20 websites had been installed.

It was decided that all files added to the codebase would reside somewhere within the sites folder, such as:

- contributed, shared modules (sites/all/modules)
- master theme (sites/all/themes)
- multisite-specific file attachments (sites/upei.ca.multisite/files)
- system-wide files (sites/all/files)
- multisite-specific themes (sites/upei.ca.multisite/themes/)

This meant that the upgrade process was greatly simplified when it came time to upgrade from Drupal 5.1 to Drupal 5.2 at the end of August. All that was needed to be copied over to the new Drupal 5.2 codebase was the sites folder.

Theme

Garland was chosen as the base theme. A variable width theme was desired, and as a core theme, chances were that Garland would be supported with later Drupal releases.

The master theme resides at /sites/all/themes/, and each multisite had their own set of theme files located at /sites/upei.ca.multisite/themes/. The multisite themes were essentially placeholders pointing back to the master them using a php include. Using this method there is no duplication of actual theme code.

Example page.tpl.php:

```
<?php
include 'sites/all/themes/upei_master_v1/page.tpl.php';
```

In the case where one multisite may require a customized version of such a php template file, the include was discarded and the full code was written into the template php file. Lastly, each multisite's sub-theme had a style_override.css style sheet that was called last, allowing any pre-existing css declaration to be overridden.

The combination of php include technique and the style_override.css technique allows for a large degree of centralized control between multisite themes, while allowing the customization of virtually any css or structural

aspect of the site.

Typical Department Site

The most typical multisite in development was the Academic Department site. These were typical public-facing websites that would be expected from an academic institution.

Their content included:

- faculty pages: photo, contact info, short biography
- staff pages: same as faculty pages
- misc static content: program info, course info, dept policies and procedures
- notices: semi-regular updates from the dept that otherwise would have been emailed out or physically posted on campus

The various levels of access to the site for administration purposes was:

- Admin assistants and communications staff manage most of the content, and were assigned the "Content Manager" Role.
- Faculty and Staff are able to manage their own pages if they wish, by changing the author info on their page to their Novell username. They were able to authenticate via LDAP as normal.
- Additional staff resources may require access to post specific content

Content types and RSS feeds

Through the use of CCK content types and views, content could be

imputted, classified and displayed much easier than if pages had to be constructed manually. Such content types were:

- Existing page content type was used to create nodes for the majority of content
- "Notice" content type was added for time-relevant items
- The webform module was used to create web forms for application forms, surveys, contact forms, etc. Their POST results had to be proxied back through to the Drupal Server using a rule in Apache on the Proxy Server.
- Events module was used to enable events to be added and displayed in a calendar
- RSS feeds were enabled for notices and events

Scalability Tools

A suite of stand-alone tools were developed to assist in managing many (100s) Drupal sites. Such tools were:

- Watchdog | to complement Drupal's existing Watchdog functionality, a page was developed that aggregated all Watchdog items from all Drupal sites. This was useful for identifying errors, and where content had been added or updated. This will be useful in the future as a certain degree of “gardening” will have to be done to maintain the professionalism of the content added by the multisite owners (spelling, grammar, punctuation, formatting)
- Variable investigation | a page was developed to identify how

Drupal variables were set in each multisite. This was useful for verifying the settings for various site attributes such as the footer, site title, site email, etc. Alternatively, all variables for one multisite could be shown.

- Cron investigation | to insure cron jobs were running correctly for each multisite, a page was developed to query each multisite for their last cron run date and time. Alternatively, a link could be clicked to run the cron job for each site centrally.
- Additional tools are currently in development.

From Development to Production

For this project, a Development Environment was created on the Drupal Server next to the Production Environment. To copy multisites from Development to Production, a script was written to do the following:

- copy the Development database to a new Production Database
- create a mysql user and grant access rights to the new Production database
- add a symlink back to the codebase from the cms.upei.ca Production docroot
- copy the Development sites folder to the Production codebase
- edit the settings.php file to reflect the new database name and credentials
- create a symlink to the files folder of the sites folder from codebase/files/multisite

From Production to Live

As described earlier, an automated process created a static copy of each multisite for delivery by the Proxy Server. The HTML files were then rsync'd to the Proxy Server, which was configured to proxy old websites through to a legacy web server, while serving up Drupal sites that had been statically cached.

The automated process happens every midnight, and Content Managers are able to trigger crawls themselves by way of a special block that appears when they are logged in.

Content porting

Over the space of five weeks, approximately 30 public websites were ported to Drupal by two permanent staff and three student interns. Only one had prior experience with Drupal.

The process of porting content was largely a copy-and-paste effort, pasting into a TinyMCE box on the CMS side. The "Tidy" button was typically used to strip excess tags from the pasted html. Menus were created and URL aliases assigned at that time as well.

Client Training

An effort was made throughout the development process to create a content management system that was accessible to non-technical staff. This was accomplished by using the TinyMCE rich text editor, which presents the user with an interface similar to Microsoft Word, Corel WordPerfect, or OpenOffice. Through this, the user is able to select from a limited number of styles that allow them to format their content without creating elements that

would be considered of bad taste. Of course no system is perfect, and the fine distinction between too much and too little control over the formatting of content is being continuously addressed.

Three staff resources outside of the CMS implementation team have been trained on the use of the new UPEI Content Management System to date, and so far the feedback and results have been positive.

Conclusion

While the UPEI CMS implementation is still within the early stages of its life cycle, Drupal has been found to be a good fit as an enterprise-grade Content Management System. Its user management and content management qualities combined with the ease of modification of Drupal core functionality was instrumental in achieving the objectives set for this project.

Conversely, it was found that the infrastructure supporting Drupal was just as important when addressing scalability issues efficiently. Through the various php and filesystem modifications, and static website copying and proxying techniques used, a content management system was built that in this case encompasses the best of many worlds: a system built to be maintainable by many, yet extremely robust, efficient and attractive. This could not have been achieved without an advanced understanding of the environment to which Drupal was running on, and Drupal itself.

About the Author

Ryan Palmer was first exposed to Drupal in 2004 when he was a student of Mark Hemphill's Management Information Science Business class. Hemphill had been leading a group working on a campus-wide weblog, Weblogs@UPEI. Palmer went on to join that group in January of 2005.

From that point, Palmer worked to develop a series of community and corporate websites using Drupal. The most significant installation since was during the summer of 2006 when a significant amount of Drupal and php work was completed for a local heating fuel delivery company.

Glossary

- Cron Job | Periodical server process run at regular intervals to perform housekeeping duties.
- Drupal | World-class, open-source Content Management System
- LAMP Stack | A standard web server environment made up of Linux, Apache, MySQL, and PHP/Python/Pearl
- Multisite | An instance or identity of Drupal. Not necessarily a stand-alone installation
- Taxonomy | The study of the classification of categorization
- WYSIWYG | “What You See Is What You Get”