Development of an Object-Oriented Module-based Extensible Student Intranet Web Application in PHP5

Andrew Deason, Eric Harmon, Bryan Rau-Jacobs, Andrew Smith

April 6th, 2006

Abstract

Intranet is the current system used for students to sign up for 8th period activities, look up information about students, and provide other school related functions. It is a web application written in PHP, and authenticates students and faculty against the school Novell system. It was designed many years ago by students, and that age is showing. It is not designed in an object-oriented approach, and is very difficult to extend or add new features to. Since new requests for Intranet are building up every day, we need a new platform for developing new features and fixing bugs. This new platform, known as Intranet2, has been nicknamed 'Iodine', which will implement several ideas in Object-Oriented programming and collaborative development. We will use a LAMP-like server setup, consisting of the Linux operating system, the Apache2 web server, a MySQL database, and the PHP5 programming language. We will also be using the Smarty template system, for increased separation of programming code and display code.

1 Goals

Our goal in the is project is to create a robust, extensible system that will grow and adapt to the needs of the school. The intent is to create a system with such flexibility that it will not need to be replaced in the near future. It should be as bug free as possible, easy to use, and optimized for speed. We

also hope that this system will be portable to any other environment with very few changes, so other schools or organizations may one day integrate this application into their network.

2 Application Structure

The majority of Intranet2 is coded in PHP5, a server-side language primarily used for websites that also supports an Object-Oriented programming model. We will also make use of XHTML, CSS, Javascript, and the Smarty template engine for displaying information.

2.1 Modular Structure

Every single line of code (except for about five short utility functions) belong to a specific module. For every intrabox that is displayed, and all content that is displayed in the primary pane, there is a module responsible for it. The module called 'core' handles all requests, parses the arguments in the URL, and passes off control to the appropriate module.

Information that is passed around is also done in an abstracted object-oriented fashion, instead of directly manipulating the database each time. In old intranet, the code to get information for a student might look something like this:

```
$addrquery=mysql_query("SELECT Address, City, State
                FROM StudentInfo
                WHERE username LIKE \"$user\"");
$arr = mysql_fetch_array($addrquery);
$address = $arr['Address'];
$city = $arr['City'];
$state = $arr['State'];
```

This uses SQL directly to access student information, so if the database structure changes at all, this code is worthless. However, in Intranet2, the code might look similar to this:

```
$usr = new User($username);
$address = $usr->address;
$city = $usr->city;
$state = $usr->state;
```

Here, if the underlying structure of the database changes, then the module 'User' must accomodate for that, but all other modules using User to get information will not have to change their code.

2.2 Automatic Loading of Modules

The special function called __autoload in PHP allows for this modular structure to be used with great ease. In the above example, the code referenced a class called 'User', which is not a normal PHP class. When the PHP interpreter discovers that the class does not exist, it calls the function __autoload (which has been defined in the application). This function includes the appropriate file with the class definition for that class (which is modules/user/user.class.php5). If the class has not been defined after calling __autload, then an error is thrown. This allows for modularization of all of our code without needing to define which modules we will actually be using. Modules are just loaded non-demand as they are needed.

2.3 Displaying Information

The system of displaying information is not simply using print and echo statements to output code, since there are several problems with that system. For instance, usually there is some code that is output on every page load, the header you see at the top of the page. However, if a module loaded after the header is displayed decides that it wants to redirect the user to another page, it can't if the header has already been displayed. This is a limitation of the HTTP protocol, but we have a solution to it.

Instead of issuing a print or echo statement when a module wants to display something, it calls the method `$disp->display()` with the information it wants to display. The `$disp` variable is an object of the Display class which is given to the module by Display itself, or by the core module. Display buffers the information that the module requested to be displayed, and only displays it at the end of processing, when we are sure that the information actually needs to be written to the display.

This has the added advantage of being able to keep track of what modules have tried to display what, and allows for an easy, flexible, template system called Smarty. Smarty is project for PHP that allows a programmer to pass variables to a template, and displays information using those variables.

The Smarty language is very simplistic, and thus forces the programmer to separate design code from processing code.

3 Usability Tests

We enlisted the help of an external tester, Chase Albert, to test how intuitive and usable the general interface was. Chase represents an average user, giving us an insight into just how well the user experience is in Iodine. Fortunately, his honesty allowed us to track down a few bugs, confusions, and ambiguities. Through this we were able to improve the average user's experience while using Iodine. Hopefully, this will allow us to reduce user error due to overly precise or technical terminology. Recent research has shown that users are often confused by systems which are designed solely by programmers, who picture their users to be at the same level as themselves. The changes as a result of these tests are reflected in the screenshots.

4 Developer Management

There were several developers working together as a team to create several sections of Iodine, so a few tools were required to organize and lead them. One of these tools is a Revision Control System (RCS), a system which allows several versions of code to be stored, and it keeps a record of who changed what and when. The particular RCS that we use is a system called Mercurial. This also allows developers to run their own instance of the application at home, or in any other environment, and allows them to work on the code without any access to the central repository.

The Intranet2 project also made use of the TJforge system, which is powered by the Trac project. Trac allows mutliple developers to contribute to a project-wide wiki system, explaining some of the larger aspects of the project. The greatest advantage of using Trac, however, is the ticket system. A 'ticket' is either a bug request, or a feature request, or some kind of ehancement to the application, or something. These allow developers to have a common place to see what there is to do, and to keep track of whom is fixing what bugs.

We also use the phpDoc system, which allows developers to document code in the code itself in the form of formatted comments. This is similar to

the JavaDoc system, but for PHP. An example might look like this:

```
/**
* Determines whether the specified user has a certain permission in
* this group.
*
* @param User $user The user for which to check the permission.
* @param string $perm Which permission to check to see if the user
* has.
* @return bool TRUE if $user has permission $perm in this group,
* FALSE otherwise.
*/
public function has_permission(User $user, $perm) {
```

This allows quality documentation to be generated from information that is created as the developers are creating the code itself.