1 Preface

This paper documents a project on web content management systems at the IT University of Copenhagen.
The project concludes second semester of the four-semester M.Sc. in Software Development and accounts for half a semester's work.
The intended audience consists of website developers and webmasters with interest in Open Source Content Management Systems.

The author has worked intensively with XOOPS for more than a year and is a part of the XOOPS Core Development Team as a core developer and leader of the module development team.

08/08/04

Jan Keller Pedersen
K-SWU
020678-1823

2 Abstract

Small and Medium Enterprises (SME's) with small budgets strive for a website, matching the websites of their competitors, but are without resources such as IT departments and website expertise and need a tool to make the process easier for them. That tool needs to be easy to use, because the website expertise is low. It needs to be flexible, so that the SME can customise the website and it needs to be low priced due to low resources in the company. These needs in usability, flexibility and price are what Open Source Content Management Systems try to cover.

In the process of this project, three Content Management System solutions were compared to the XOOPS Content Management System in order to locate an area of XOOPS that could be enhanced to improve the product as a whole.

The comparison showed that XOOPS is a capable system with good flexibility, but also a system, which has existed for a while with legacy code and backwards compatibility interfering with the easy understanding of the features available. Also, the XOOPS module architecture does not facilitate the interaction between modules.

Based on the comparison, it was decided to analyse, design and implement a hook system, allowing modules to interact with each other dynamically.

The implemented code allows for modules to raise and respond to events and is configurable in the administration area, however the solution is not ready for production until additional functionality is added to the package.

Table of Contents

3 Introduction

When the Internet began to take off for commercial use in the mid-90's, it was mainly used for static brochure-like pages giving information about the company. After a while, the need for dynamically updating the websites rose, as the companies recognised the demand for up-to-date information and this evolved into sites with visitor interaction, first through feedback forms and later through elements such as support forums. Maintaining this dynamic, eternally evolving, website calls for multidisciplinary content management: People, who have the knowledge supply to maintain the content and the tools to do so. Webmasters have gone from the role of maintaining all parts of a website to a support role, supplying tools for others to maintain the content. Today, instead of having central webmasters, there is more a need for defined roles in maintaining the content, layout and appearance.

Small and Medium Enterprises (SME's) with small budgets strive for a website, matching the websites of their competitors, while differentiating themselves from them. However, SME's are often without resources such as IT departments and website expertise and need a tool to make the process easier for them. That tool needs to be easy to use, because the website expertise is low. It needs to be flexible, so that the SME can customise the website and it needs to be low priced due to low resources in the company. These needs in usability, flexibility and price are what Content Management Systems (CMS) try to cover.

When talking about affordability, a logical step to take is to look at Open Source software. Open Source software is characterised not only by a low price – if not completely free – but also by definition access to the source code of the software. Large proprietary systems such as MicroSoft SharePoint, often have areas that are closed for the buyer and only a limited amount of customisation is possible – despite a high acquiring fee. Also, it is not uncommon that proprietary systems require top-class databases, adding to the acquiring costs. Open Source systems usually work with smaller and cheaper databases, grant full access to all parts of the source code and hence it is possible to customise it as much or as little as you want – provided you have the knowledge to do so.

An Open Source system can be compared to the city bike project in Copenhagen, where simple bikes without gearing etc. are made available to the public for a small deposit. When the goal is transportation through the city, these city bikes can be a good, inexpensive way of learning whether there is an advantage of having a bike, whether there is a need for a bike to yourself and whether there is a need for gearing etc. which you can get when buying your own bike.

An added benefit of the Open Source systems is that while learning about e.g. MS SharePoint will give the IT staff competencies when working with this software, it is close to useless when working with other systems, whereas the availability of the source code in Open Source can be a learning ground for the staff to gain knowledge that can be applied in another system. Buying a proprietary system and learning how to use it can tie you firmly to that system, whereas the switching costs between Open Source systems can be significantly lower.

3.1 Problem Statement

This comparison will focus on the needs and resources in SME's in order to evaluate the Open Source content management solutions available to them. The main tasks are to investigate the following questions:

1. Which content management solutions are available to SME's to use in their company website?
2. How does XOOPS compare to the other solutions? Is XOOPS a viable solution to use?
3. Where can XOOPS improve to become a more competitive product?
4. How could this improvement be implemented?

3.2 Structure

The XOOPS content management system will be compared to other systems, exploring the possibilities and potential in these systems, by evaluating the system from a user as well as webmaster and developer viewpoint.

The result of the survey is highlighting the areas, where XOOPS is strong as well as areas in which XOOPS is lacking. These areas will be the basis of requirements to improve XOOPS in order to enhance the system from a product perspective. In the last section, a solution to the requirements will be analysed, designed and implemented.

To avoid conflicts of interests, the criteria for comparison will be as objective as possible and since the goal of the comparison is not to declare a winner, but to improve XOOPS, there will not be a need for shortcuts placing XOOPS in a more favourable light than it deserves.

4 Content Management Systems

A Content Management System (CMS) is a broadly defined term. www.whatis.com defines it as “a system used to manage the content of a web site [...] without needing the expertise of a webmaster“ but the predicate is put on systems ranging from weblogs (online diary-like columns where registered users can write about what goes on in their lives) to community site software to full-blown content management systems used to control every piece of information available on a website. Commonly, a CMS provides a framework for site content, navigation and user authentication.

Common for most multi-purpose CMS's (excluding CMS's focusing on specific areas such as weblogs or course management) is that they have a modular structure with the possibility to add modules, extending the functionality of the system.

Another common concept in CMS's is a “block”. A block is a small piece of code, displaying a limited amount of information, such as the latest articles or forum threads. Blocks can normally be placed on any page, thus highlighting an area different from the viewed page.

Since the visual aspect of a website is very important, most CMS's separate presentation logic from application logic by supplying a way to make themes and templates. Themes control the general positioning of various elements such as banners, blocks and module content and the colouring used in the various elements, whereas a template controls the layout of the individual elements.

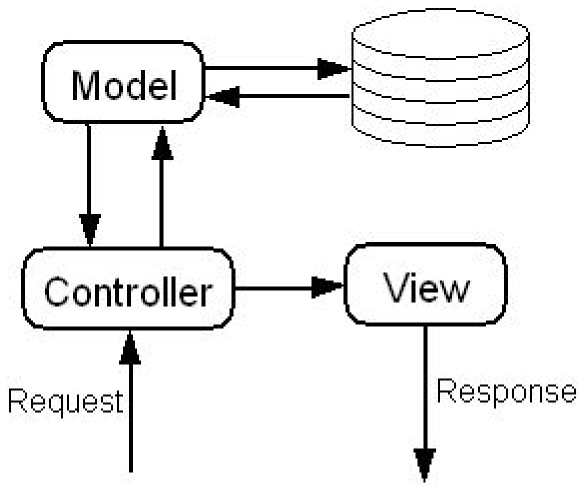
For more definitions and terminology, please consult Appendix A: Terminology.

4.1 Model-View-Controller

Most CMS's make use of the Model-View-Controller paradigm – whether consciously or not.
The MVC paradigm is a way of breaking an application, or even just a part of an application, into three layers: the Model, the View, and the Controller. The objective of this separation is to make it simpler to modify only one part of the system without fear of breaking other parts.

The Model is the processing part of the application – such as database queries and application objects – the View is the various screens being displayed and the Controller is a collection of files receiving user input, requesting model processing and selecting the view to display on the page. The

MVC relationship is illustrated in the figure below:



Following this model in a web environment, the three parts of Model-View-Controller can be:

1. A file (Controller) is called through the browser address field (Request) – possibly with additional parameters
2. The file process the parameters if applicable and include classes or functions (Model) to process the information that needs to be displayed or retrieve it in the database, and arranges it in a suitable way for the View part to understand it
3. When the processing is done, the file includes the appropriate code for positioning the information on the page (View) and sends it to the browser (Response)

5 System Selection

There is a large number of CMS's available; CMS Matrix (http://www.cmsmatrix.org/matrix) lists 56 CMS's and Open Source CMS (www.opensourcecms.com) lists 33 (some intersection between these two sites).

The forums and documentation available on the mentioned sites were used in order to end up with a manageable number of CMS's, and the 10-15 most promising systems were downloaded and investigated. The following criteria were applied to find the most suitable CMS's for Small and Medium Enterprises (SME'S):

1. *Root access to the server is not required for installation* - SME's on a limited budget will most likely use a hosting service to hold the website. These hosting services are very likely to share multiple websites on one server and will not give root access to the server to their customers.
2. *System must be internationalized* - SME's can be located anywhere and to give the system a broad appeal, the system should be adaptable to other languages.
3. *System must target non-technical segments* - SME's with small budgets and IT competencies should be able to use the system without too much trouble. This is hard to estimate, but a relative comparison among the selected CMS's can end up in one or more CMS's being excluded on this criteria.
4. *System must be suitable for a company website* - There are several excellent CMS's focusing on a limited area such as a weblog or course management for universities and schools. However, these systems are not very useful for a small company, positioning itself on the Internet and will not be taken into consideration.
5. *System must have an easy installation of modules* - For an inexperienced webmaster, it is crucial that it is easy to add modules with specific functionality to give the website the content it needs, whether it is for an Intranet, a public face for the company or a support site – or a combination of the three. If a site is created for one specific purpose, it must be easy to expand the site to cater for new needs as and when they arise.

5.1 Resource Limitations

There are limitations to how many systems, I can review as I am the only reviewer and have a limited period to do the comparison in. Therefore I find it necessary to add additional filters to ensure that the best systems are compared in a fair comparison of each CMS with extensive code-diving and understanding.

1. *CMS written in PHP* - This comparison will feature heavy code-diving and therefore it must be comparable on this level – and since the author's competencies lie in PHP, it would not be fair to compare with a Python or Java CMS.
   From a SME viewpoint, PHP has also the advantage that it is very common in relatively cheap webhosting packages, as well as it is quite easy to setup, should the SME decide to host the website themselves.
2. *System must distinguish itself from the rest* - There is no sense in comparing several similar systems, since the objective of the survey is not to find the best CMS available, but to get an overview of the various features available in CMS's. Each CMS should bring something of value to a SME to the equation.

5.2 Candidate Systems

Filtering systems by the criteria and limitations, the following systems were investigated further:

1. phpNuke
2. PostNuke
3. XOOPS
4. e107
5. Mambo
6. Xaraya
7. Drupal
8. eZ Publish

Each CMS was downloaded and installed on a test system with a 1 GHz Athlon processor and 256 MB RAM.
The test server is running Linux Mandrake 10.0 Community operating system, a MySQL v4.0.18 database and an Apache 2.0.48 webserver with PHP 4.3.4 and XML module.

Looking at the requirements for the candidate systems, they commonly require PHP version 4.1, MySQL 3.x and Apache 1.3. Seeing as the test set-up fulfils these requirements it is deemed usable for the comparison.

5.2.1 PhpNuke

At first there was nothing, then came phpNuke. That is more or less the story of phpNuke, which started in the summer of 2000.

PhpNuke is written in mainly structured PHP and as the "father" of almost all PHP Content management systems, phpNuke is a strong contender to a place in the comparison, but some elements count against it. The main element is that since many of the other CMS's are based on phpNuke, the code will most likely be resembled in the other systems. The bug tracking and fixing has also been very erratic and even though there are support forums to turn to with problems no less than 14 of the latest 25 bug reports were not even replied to, when accessed on May 12th 2004.
Since phpNuke is also known as a system with an enormous amount of hacks and many forks – also internally in the project, where some modules only run on a certain version of phpNuke with a certain hack applied – this CMS is really not suitable for a comparison.

5.2.2 XOOPS

XOOPS is an abbreviation of eXtensible Object Oriented Portal System and is a fork of PostNuke, but the core has been completely rewritten to use object orientation wherever possible. XOOPS has been developed since December 2001 and is currently in version 2.0.6.

5.2.3 e107

e107 was started in September 2001 and is currently in version 0.614. There is some object oriented code, but most is done through structured PHP and the use of functions. Themes are made with PHP code. Installation was easy, but it is not easy to understand the administration menu and how to install additional modules. I did not succeed in installing an additional module after installation and hence the system falls for the limitation that "System must be easy to install modules for"

5.2.4 Mambo

Mambo is positioned as a very easy and user-friendly CMS. Existing since 2001, Mambo is currently in version 4.5 with a 4.6 version in the near future.
Similar to XOOPS, Mambo also has a very object oriented kernel.

5.2.5 Xaraya

Xaraya is based on PostNuke, but with a major overhaul of the code and architecture in order to make a CMS, focused on managing community sites. The Xaraya developers work hard on a modular architecture where as little as possible is placed in the core. This shows in the directory structure, where very few files reside in the Xaraya root directory.
Xaraya is currently in beta development with a version 0.9.9, but approaching a full release.

5.2.6 Drupal

Drupal is not based on any previous CMS. The version evaluated is Drupal version 4.4.0, available on www.drupal.org – a very clean site with a good overview of what information is available there. The Drupal organisation also offers advertising space to webhosting companies and consultants, working with Drupal.

5.2.7 eZ Publish

eZ Publish is a commercial solution targeted at corporate websites. Per default, eZ Publish can setup a forum site, a corporate website or an Intranet. Quite a number of files and folders need to be writeable by the server prior to installation and some settings are needed in the webserver setup, but apart from that, the system is easily installed. However, once installed, there is little possibility of installing additional modules. As an example it was impossible for me to install a support forum on an "Intranet" eZ Publish site.

5.2.8 PostNuke

PostNuke is a fork from phpNuke, started in May 2001. The PostNuke developers have as goal to provide the same level of features as phpNuke, but with less code, less load on the web server and with a more modular architecture.
PostNuke is currently going through major changes in the structure and architecture, mainly in the template system and it is not deemed stable enough – that is, the code may change substantially – for a direct comparison. Both XOOPS and Xaraya are based on PostNuke and it does not bring anything new that these two systems do not have.

5.2.9 Selected Systems

The following systems were found to fulfil the criteria and limitations:

1. Drupal
2. Mambo
3. Xaraya
4. XOOPS

5.3 Categories for comparison

Each CMS will be compared on the following four categories, which I believe will cover all vital aspects of the system.

1. *Installation* – As previously stated, the assumption is that a SME will not have skilled IT administrators so the installation procedure must be suitable for this situation.
2. *Documentation* – The SME is assumed to desire to customise the system, so the documentation describing how to do this should be evaluated. The level of documentation will

be investigated and will cover everything from installation instructions over user and administration documents to developer documentation explaining how the CMS is structured, features available in the core and how to use them.

3. *Usage & Administration* – Staff at a SME should be able to setup the CMS without professional help.
4. *Development* - SME's are assumed not to be interested in paying high consulting fees for minor changes or custom features. The easier it is for the SME or a consultant to understand the system, the easier it is to develop for and the smaller a consultant fee will be. This demands that the CMS is easy to understand and that the functionality in the core can be used in modules to make tedious tasks such as user authentication and access restrictions easier.

6 Comparison

6.1 Xaraya

6.1.1 Installation process

Installation is done through an installer script that will create the database if it does not exist, create and populate database tables and let the user select the modules, he or she wants to install during the installation process.
During installation, a list on the right side shows the progress. Files and folders needing to be writeable by the server are listed with explanation as to how to do it.

After setting database settings – such as database location, username and password - and creating an administrator account, the installer will ask which kind of site is wanted from the options "Community Site", "Core Xaraya Install" (a minimal installation), "Intranet" and "Public Site". The choice made will affect the options checked on the next page, where the user can modify the selection to install the modules, he or she desires.

All in all a very clean installation with the most important settings and configurations set during the process as well as installation of modules – which can all be changed subsequently in Xaraya's administration.

6.1.2 Documentation

Xaraya is shipped with a text file containing installation and upgrading instructions and information on basic usage. Each of the standard modules are explained as well as the core functionality – which is also very modular. The text file format is not very easy to read, but ensures that it is readable by just about every computer available.

The Xaraya community has contributed to a quite extensive guide for module development, outlining how to create modules and use the core functionality. This 44 pages manual explains modules in relation to Xaraya in great detail with code examples and good headlines. All in all an excellent piece of documentation.

Only drawback is that it passes reference to an API guide to Xaraya that is yet unavailable, but since Xaraya is still in beta this was not unexpected.

6.1.3 Usage & administration

After installation, it is possible to login with the account created during installation. The modules specified during installation are available in a main menu, some with sample items already there. Administration options are available in an admin menu, where blocks are created, modules installed and configured and users administered.
The administration menu is divided into categories; Global (blocks, mail setup, modules and themes), Content (Individual modules), Users & Groups (usergroups and access rights for each usergroup) and Miscellaneous.

Each element in the administration menu has a front page, explaining how to use this element. Unfortunately, this is very often necessary because of a highly modular architecture and very flexible functionality, which requires a high level of configuration.

Administration

A lot of fine-tuning is possible in Xaraya, but it is not very easy to understand how for example the access permissions work. The webmaster needs to create individual “privileges” such as read access on a block and then assign it to a user or a usergroup. Some pre-defined privileges exist and are assigned to the Administrator group on installation, but it is not apparent what permissions like “LockGeneralLock” privilege do and why they are there.

Hooks

Xaraya has an extensive “hook” system, enabling a module to register hooks with Xaraya, which can then be connected to module items. For instance, a “comments” hook can be connected to the “articles” module, enabling the users to post comments to individual articles. The hook implementation will be further investigated in the Development section.

Blocks

Blocks are positioned in various areas of the screen. These areas are

- Admin – left side, admins only
- Header – in between the <head> and </head> tags of the HTML page
- Left – Left side of screen
- Right – Right side of screen
- Center – Center column of screen
- Topnav – Horizontal line above the other block areas

A Block is created as an instance of a certain block type that is supplied by a module. This means that an indefinite amount of instances of the same blocks can be created, opening up the possibility of having a "recent news" block visible in the left block area on some pages and in the right block area on other pages. Alternatively, two instances can be configured to show different items – e.g. "newest articles" and "articles with most reads".

Themes

From the administration menu, the webmaster can select the theme to use on the site as well as configure various parameters such as copyright notice and site name, which are displayed on every page on the site. It is however quite confusing that there are six themes available by default, whereas only two can be selected as the site theme. Due to the theme names, I assume that the "print" theme is used on printing pages and the "RSS" theme on RSS feeds – but it is not explained why e.g. the "Atom" theme is not selectable as site theme.

6.1.4 Development

When looking into the code, the heritage from phpNuke shows – but quite some enhancements are noticed right away, too. The heritage shows in that it is always the index.php in Xaraya's root that is invoked with parameters to include module files. This gives URL's with a very long query string (arguments after the host name, directory names and file name) and clouds which file is actually invoked.
On the positive side, the modularity shows already in the Xaraya root as there are only 5 files; 2 for installation/upgrade, index.php, val.php for validation and ws.php for an interface to SOAP or XMLRPC (web services interfaces)

All core functionality is either separated into modules or collected in the "include" folder, which can be compared to a functions library, available to any module.

The Xaraya architecture is somewhat structured, but with aspects of object orientation. An object oriented style in modules is not discouraged, but warned that it must follow the structure of Xaraya's function and file naming scheme.

Modules

The heritage from phpNuke also shows in the structure of module files. The module MUST follow a rigid file and function naming scheme. This is in order to use the Xaraya Module API to call functions with a single function and parameters. E.g. xarModAPIFunc('articles','user','getpubtypes');

will call the articles_userapi_getpubtypes function located in modules/articles/userapi/getpubtypes.php.

There are advantages and disadvantages of such a rigid naming structure. It makes the code very hard to read, when all function calls are to core functions with the actual functions desired in the parameters and when each module function must be in its own file, it creates an immense amount of files in a module. However it truly divides the MVC parts of the Model-View-Controller design pattern by dividing functions into API functions (Model), templates (View) and user/admin files (Controller).

Hooks

A hook in Xaraya is a function to be run, when triggered by an external event. It could be the ratings module, which has a hook function to delete all ratings from a certain module, when that module is uninstalled.

Hook functions are registered in the initiation function in the xarinit.php file that is run on module installation. Modules, where the hooks should be "attached", will need to specify this in *their* xarinit.php file with a core function call for each hook module. Modules attaching hook functionality will also need to add code to their templates to specify where the output of hook functions should be positioned.

Themes

Themes in Xaraya are XML documents defining areas for blocks and module output and filling in with values from the PHP files. The real layout and colour definition is in various CSS files in the theme through the use of CSS classes and ids in the XML document.

Debugging

In Xaraya the webmaster can enable debugging output, which is a dump of all variables on a page. However, this needs a bit of work, since it when enabled outputs a message along with the debug information about the module breaking Xaraya coding standards.

Core Features

The main core features in Xaraya available to modules to use are exception handling, text sanitation and functions for building URLs. Apart from that, functionality is mainly added to a module via hooks.

6.1.5 Conclusion on Xaraya

Xaraya is a good system, I think. It is the most modular PHP CMS in this comparison. Hooks, blocks and permissions can be configured completely by the webmaster, which adds an extreme level of flexibility, but also adds considerably to the complexity. Xaraya has a very dynamic and impressively flexible permissions system, but it is very difficult to understand and takes quite some time to set up properly.

Even though the installer works great and will install modules from the beginning, it cannot be recommended to persons completely new to the CMS world as it IS quite complicated to set up properly – even with the help of the otherwise excellent documentation.

When digging into the code, the complexity needed for the flexibility shows again with very complicated code reading due to rigid function naming and placement, one central index.php used for accessing all pages and very low object orientation in the system.

The phrase “easy to learn, hard to master” comes to mind when using and administering a site with Xaraya. It is quite easy to set up a working site, but hard to configure because of numerous toggles.

6.2 Drupal

6.2.1 Installation Process

Drupal is packaged without an installer script, but with a text file explaining how to set up the system. A database interface is necessary in order to create the database and populate it with tables and data. Also, it is necessary to manually edit a configuration file to hold the database location, username and password.

6.2.2 Documentation

Along with the Drupal source code is also an html document explaining the basics of Drupal. This document seems very important because of two special terms in understanding Drupal: “Taxonomy” and “Nodes”.
Generally, taxonomy is for grouping content together in categories. A node is a piece of content, whether it is a news article, a forum post or a calendar item. One could e.g. have a taxonomy term called News with subcategories Sports, International and Domestic as well as an Event term with the same subcategories. It should then be possible to show all Sports news articles and calendar

events through the use of this taxonomy.

The documentation is thorough, but the impression is clouded somewhat by the fact that it also includes user comments with unanswered questions. This does not give the impression that it is a very competent piece of documentation.

6.2.3 Usage and administration

Administration

The taxonomy is a very integrated part of Drupal with every piece of content being a node and grouped in a taxonomy term. This also shows in the administration, which is divided into the following groups:

- Content – Containing the nodes created on the site and allows to group some of them together in an inter-category "book" collection of nodes
- Comments – Comments is a core feature, which can be added to nodes and managed from this area, where comment moderation can also be configured.
- Accounts – User administration and permissions. A user is assigned a "role" for which permissions are set. Permissions can be e.g. "administer comments", "create forum posts" or "access content"
- Configuration – General settings such as caching, timezone settings and a site-wide footer. This area is also where modules, themes and blocks are configured.
- Taxonomy – Now we get to the crucial part of Drupal. The Taxonomy item is where the various taxonomy terms are created and related to each other.
- Logs – All actions are logged in here, be it nodes being created, error messages encountered or failed logins. Each item is coloured according to its group – e.g. php errors and warnings are coloured red, page not found errors coloured green and node creation coloured yellow

Navigation

Navigation in Drupal is centred around two items: The navigation block and primary/secondary links. The navigation block holds an item called "Create Content" which will show a page with all the nodes available to create – such as news items and forum posts – but it seems that the real navigation is done through blocks showing e.g. the most recent forum postings, and through the primary/secondary links which are two lines of links, configured in the administration area as hardcoded links to various parts of the site.

Blocks

Drupal has a very simple block configuration. There are two block placement areas: Right and left. Blocks can be either on or off, but they can also be flagged as "custom", allowing a registered user to turn off the blocks, he or she is not interested in. However, it does not seem to be possible to set a block visible to certain users only.

Themes

Drupal is shipped with 3 themes per standard which can be enabled/disabled in the administration area as well as the default theme can be set. This worked flawlessly and as expected.

6.2.4 Development

Drupal is written almost completely in structured PHP with heavy use of functions. As with Xaraya, a single index.php file controls the information displayed and there are very few files in the root folder. Drupal use a number of functions for calling other functions based on parameters.

Modules are located as single files in the modules folder containing all the functions used by the module to retrieve and save data in the database. The "core" files containing basic functions are in the include folder, all parts outputting information are built as modules, with the "Node" and "Taxonomy" modules being the central modules and required in order to run Drupal.

Modules

As mentioned all modules in Drupal consist of just one file. This file must contain a number of rigid named functions such as modulename_page for retrieving the information used on the page and calling the core function theme() with parameters type and content. The type can be "page", "block", "box", "comment", "node" or "help" which corresponds to functions in the theme, executing code accordingly to display a page, a block etc.

Themes

The default theme uses XTemplate, which is a PHP-based templating system and should help divide presentation from business logic, but it does not really give the impression that anyone without PHP programming expertise could just pick this up and use it as there are PHP functions for processing intertwined with presentation directly in the theme.

Debugging

Apart from the logging of errors, there is no systematic debugging in Drupal.

Core Features

Drupal includes functions to aide file uploading, sorting table rows, redirecting to another page and a collection of functions to make form creation easy.

6.2.5 Conclusion on Drupal

Drupal is a very interesting system. It has adopted the paradigm that content can be standardized – especially since it does not seem to be possible for modules to add to the database structure. This, however, results in modules outputting code with a very uniform look. E.g. it is practically impossible to figure out just by looking at the page if a node displayed is a news item or a forum post as they look exactly alike.

Drupal has, just as Xaraya, a considerable flexibility, but it is clouded by the fact that it is very difficult to put your mind around the taxonomy and node terms. Once grasped, the idea is indeed very nice, but this system is also a webmaster's nightmare due to the heavy configuration needed in order to get a site looking like you want it to with the modules and blocks you want as well.

The heavy reliance on rigidly named functions and core functions to call other function based on parameters makes it hard to read the code and difficult to figure out where the output is generated – but this is also a side-effect by the taxonomy/node principle.

6.3 Mambo

6.3.1 Installation Process

Installation of Mambo is a breeze. The installer script works flawlessly and is just as simple as the Xaraya installation.
If the database does not exist, it will be created and any files and folders requiring to be writeable by the server will be listed. Quite a large number of files and folders need to be writeable, though.

6.3.2 Documentation

On the official Mambo website, an official manual is available. This 102 pages document explains how Mambo works in great detail, however it is focused on the usage of Mambo and the official modules (which in Mambo lingo is called “Components” - “Modules” in Mambo is what was earlier defined as Blocks). The manual contains not one line of code, but it is expected to come in later versions of the manual, which will contain an API reference guide and a Developer's Guide.

6.3.3 Usage and Administration

Mambo is a very nice looking CMS. When it is loaded up for the first time, the site looks much less "block"-like than some other CMS's, meaning that the blocks are an integrated part of the website look and not looking like independent areas.

Users are granted permissions based on a hierarchy going from "Registered" over "Author" to "Editor" and "Publisher" with increasing permissions. Users are given access to modules and blocks through three levels: Public (Anonymous and registered users), Registered (only registered users) and "Special". These "Special" users can also be permitted to submit news and add polls – unfortunately the manual is very limited as to how to *set* a user's status to "Special" and my experiments did not succeed in doing so.

Administration

The Mambo administration has a separate login area with separate administrative users. Mambo succeeds in making a very clean and intuitive administration section. JavaScript menus divide the administration into functional units. Everything is done to make the navigation as small as possible, leaving a large area to getting an overview of the various settings, modules and blocks.

Navigation

Navigating through modules are done with the main menu, where links to the various modules are placed. Content is added via links in the module content area or by using links in the user menu. Both menus are customisable in the administration area.

Blocks

Blocks are defined by the modules and can be either published (visible) or unpublished (not visible) to either all users, registered users only or "Special" users only.

Themes

As with the rest of the administration area, finding the themes (called "Templates" in Mambo) is easy. A theme can be set as default and the rest selected by the users through a theme selection block. A nice touch is that the theme HTML and CSS code can be edited – or a new theme created – directly from the administration.

6.3.4 Development

Mambo is mostly written in object oriented PHP with a central index.php controlling the output. Modules are in the components folder, blocks in the modules folder and administration files are in the administration folder. Apart from that, there are core files in various folders depending on function and structure, e.g. the classes are in the class folder and helper functions in the include folder. All in all a very clean structure that compensates for the lost overview, caused by the central index.php file.

Modules

Modules need to have a file in the components/com_modulename folder called modulename.php in order to work. It should also have a template file called modulename.html.php and can include a modulename.class.php for placing module-specific classes.
Mambo strives with this structure to separate presentation logic from business logic following the MVC design pattern.

The only thing speaking against Mambo's file and folder structure is that if a module has blocks and administration, then it will need files placed in administration/modulename, components/com_modulename and modules/modulename, making it a little inconvenient to install and uninstall modules and blocks.

Themes

Themes in Mambo are made with PHP with calls to functions for building blocks. A peculiarity is that the permission check for viewing a page is done with an inclusion of the root file mainbody.php directly in the theme and the theme is therefore not the last element to be invoked. Checking permissions first and applying the theme last is the normal flow in the other reviewed CMS's.

Debugging

In the administration menu, debugging can be enabled. This will open a popup window listing all variables set on the page with their values. However, this seems to only work in the administration area and is without effect in the live site.
Mambo can be set to four levels of error reporting: Default, None, Simple and Maximum. Default setting will use the default set in the webserver's PHP configuration, None will turn off debug, Simple will show some and Maximum will show all PHP errors on a page.

Core Features

Mambo's core library contain classes for PDF file creation, date manipulation and also a form element library with a WYSIWYG editor (What You See Is What You Get – i.e. A text editor that can change the text appearance as it is written instead of having special or HTML tags before and after text to format it)

6.3.5 Conclusion on Mambo

Mambo is a very good-looking system. Everything is displayed in a good way with some excellent themes and a stunning administration interface. However, some things are not perfect in this CMS; Mambo is meant for a public site, mainly distinguishing between anonymous and registered users and not geared for more granular access control. Administration and navigation is easy and straight forward and the manual is very thorough when it comes to using a standard installation. However, structuring module code for Mambo is not as straight forward. Just as there is one central index.php in the Mambo root, all module actions must be carried out through a single file in the components/modulename folder. Module files are split between directories for administration, user accessible files and block files, making it less simple for the webmaster to install and remove modules.

Mambo is a good system for the user and administrator once modules are installed, but less flexible than the others for the developer.

6.4 XOOPS

6.4.1 Installation Process

XOOPS installs a basic package without problems. 3 folders and one file must be writeable by the server and if the database does not exist, it is created, provided the database user has rights to do so. During installation an administrator user is created, who will be able to log in right away. Just like with the previous CMS's.

6.4.2 Documentation

Documentation for XOOPS is a lacking point. The most necessary information to install is included as an HTML document, showing the screens a user must go through in order to install XOOPS, but apart from that, there is no documentation in the package. On www.xoops.org, a “Visual Introduction to XOOPS” is available that presents XOOPS, the official modules and a range of websites, using XOOPS. For using or troubleshooting XOOPS, information is available on the www.xoops.org website, outlining the most important aspects of setting up a XOOPS site and developing modules and themes.
There are quite some holes in the documentation and a considerable amount of trial and error is required in order to fully understand XOOPS.

6.4.3 Usage and Administration

After installation, it is possible to log in with the administrator user and via the “Administration Menu” item in the main menu gain access to the site administration. By default no modules are installed and it is necessary to go to the administration menu to do so.

Administration

The administration menu is divided into modules with the core system being the System Admin module. This module configures settings for usergroups, blocks, banner ads, images, users, templates, modules and general settings such as site name, meta tags and email settings.

In groups admin, the various usergroups are created, members of the groups edited and access to modules and blocks configured. Groups cannot inherit from another group, but users can be members of several groups, cumulating their access and permissions to modules and blocks.

Blocks

Blocks are administered in the Blocks admin, where available blocks are listed and can be set visible or invisible as well as ordered by weight. Five areas are available for positioning blocks: Left, Right, Center-Center (full width between right and left column), Center-Left (left half of center area) and Center-Right (right half of center area, next to the Center-Left blocks). Each block can be set to be visible on all pages, certain module pages only or top page only. The top page is configured in the general settings and can either be an empty page with only top page blocks or a module page. If set to a module page, it will show module contents, blocks to be shown in the module and top page blocks on the site's front page.

Themes

Available themes are listed in the general preferences, where one theme is selected as default and it can be specified, which themes can be selected by the users in a block, if this block is set to be visible.

Templates

XOOPS distinguishes between themes and templates. Themes decide the general layout and colouring for the page whereas templates dictate the layout of the detailed content such as a forum post or a news item. Themes are not editable, but templates can be edited directly from the administration menu. This is a really nice touch as it opens up for individual customisation of content layout by webmasters.

Navigation

The main menu is populated with links to modules available to the user unless a module has been configured in modules admin to not display a link there. Apart from this, the main menu cannot be customised in the site administration. Each module can specify sub-menu links to different pages in the module, e.g. the News module has a main link to the news front page and a sub-menu link to the archive, where news are grouped by month and year. These sub-menu links are not editable in the administration area.

User Menu

A logged-in user has also a user menu where he or she can edit his or her user account, see the inbox of the built-in Private Messaging system or get an overview of items that the user subscribes to, using a built-in notification system that can send a private message or an email when certain events occur.

6.4.4 Development

XOOPS is mainly written in object oriented PHP with module files placed in subfolders to the "modules" folder. Blocks are integrated with modules and located in subfolders to the module's folder. Administration files are placed in the modules/modulename/admin folder.
Instead of a central index.php, pages are loaded through accessing individual files that include XOOPS-specific files in order to be displayed within the XOOPS framework.
XOOPS is however not a new system and the legacy from previous versions is still visible in some areas, e.g. there are several functions for presentation that are no longer used as well as some classes with methods which are deprecated but still present for backwards compatibility.

Modules

There are no restrictions to file names apart from an admin/index.php file for navigation in the administration menu and a file in the module's root folder called xoops_version.php, that contains the configuration of the module's blocks and templates and specify use of XOOPS core functionality such as the notification system or comments system.

Themes

XOOPS themes use Smarty Templates, a templating system developed by a 3rd party. Smarty Templates are written in mostly HTML with some specific tags, which can be used to perform simple actions such as going through each element of an array or have if-else clauses. The idea of Smarty is to have as much presentation (HTML) as possible and only use programming (Smarty

tags) where it is necessary in order to present dynamic data. Variables can be “assigned” to a special object in the regular PHP code and subsequently accessed in the templates.
When a Smarty template is first accessed, the special Smarty tags are compiled to normal PHP code and placed in a file in the templates_c folder, from where it will be retrieved when accessed later. Smarty furthermore allows for caching template output to reduce the load on the database. XOOPS allows individual blocks and module pages to be cached.

Debugging

Per default, XOOPS blocks all error messages to the users, but they can be turned on in the administration menu. XOOPS also has a debug popup window for debugging the database queries, where every query performed by XOOPS on a page is listed – failed queries marked in red along with an error message – and Smarty has a debug option, also configurable in the administration menu, which will show all variables assigned to Smarty and their values, similar to the Mambo debugging option. Together, these three debug levels aid in finding the area that is causing problems. The only bad thing to say about XOOPS debugging is that it is not possible to enable all three debug levels at once and that it takes just a little too many page views to change the settings.

Core Features

XOOPS relies more on built-in functionality than the other reviewed systems. As mentioned earlier, XOOPS has a comment and notification system, which is implemented as hooks in the other systems. Apart from that, XOOPS has a class library with classes for various form elements, file uploading, text sanitation, emailing and private messaging. In addition to these elements used in the module files, modules can specify in the xoops_version.php file, which configuration elements, they have. E.g. a module can have two different ways to display an overview page or let the webmaster decide how many items are displayed per page. With a few lines in the xoops_version.php, this configuration setting is automatically added to a preferences page in the module's administration where the webmaster can give the setting a value. The setting is saved in the database and usable in the module files.

6.4.5 Conclusion on XOOPS

XOOPS is a capable system. Easy to install, reasonably easy to use, but limited in documentation and not as visually stunning as Mambo.

The administration holds the functionality necessary to configure a site, with different access levels for different usergroups without too many problems. Features like comments and notifications are

available for modules, but not as pluggable as in Xaraya.
In general, XOOPS is a flexible system that is not as restrictive as other CMS's, but also shows that it is not a new system with a considerable amount of legacy code and multiple inconveniences due to back-compatibility.

The XOOPS core functionality is not as sharply divided from module functionality as the other CMS's, which makes it somewhat harder to exchange one area of the core with a new and improved part.

7 Comparison Conclusion

We have looked at four very capable CMS's and luckily the aim is not to name the best CMS – because that would be very difficult. Each CMS has its strengths and weaknesses and a different approach to content management.

Xaraya aims at giving the developers a very modular tool to make web applications as well as a highly configurable tool for webmasters to manage their website. Xaraya has excellent documentation, but it does not take away the impression of a complicated system, which is not easy to work with before it is fully understood. However, it has great potential with modules using hooks extensively and dynamic data being attached to just about any object.

Drupal is also highly modular with the most special features being the standardized content and the Taxonomy paradigm. As with Xaraya, this does make it difficult to learn to use Drupal optimally and I am by no means considering myself an expert after this survey. There is much to learn about Drupal – both for developers, webmasters and users.

Mambo on the other hand is very easy to use. It looks stunning and both user side and administration is well structured and easy to comprehend. The object oriented code makes it easier to read than the two first CMS's, but a module is spread over up to three folders and the file structure itself does not look as if it is meant for comprehensible modules with complex functionality.

XOOPS shows its legacy from PostNuke, but with a more object oriented kernel than the other CMS's and with a great deal of flexibility in module structure, it is easy to develop for XOOPS. The navigation and configuration is not as good looking as Mambo's, but is functional without being as complex as Drupal and Xaraya. The framework allows for just about any working standalone PHP script to be ported to a XOOPS module with only a change in functionality regarding user authentication and database connectivity required. This, however, is also a drawback as it allows modules to be very unstructured and "ugly", whereas a port to Xaraya, Drupal or Mambo would require the entire structure to be rewritten, making the task bigger, but the end product nicer and more in line with the CMS.

8 Feature Development

8.1 XOOPS Feature Requirement

The CMS comparison showed that XOOPS does not have a hook system and modules are perceived as individual entities without interaction with one another. Comments and notifications can be attached to items, but this is a core functionality that a module can make use of and not easy to exchange for better functionality. Other CMS's have comments as a hook, which can be attached to any item in a module. A hook system in XOOPS would enable XOOPS to offer the same as other CMS's while keeping the flexibility and the object oriented API, thus gaining a competitive advantage.

The following section will document the analysis, design and implementation of an object oriented hook system for XOOPS, attaching extra functionality to a module item as well as investigate the possibilities of a similar interface to perform actions in one module based on events in another module.

The objective of the development is a hook system for XOOPS. To illustrate its use in XOOPS, a ratings module will be developed and a small module will be created to take advantage of this hook.

8.2 Development Approach

The development will be carried out in three iterations, each with individual planning, analysis and design as well as coding and test:

1. First Iteration: Basic structure. When a page is displayed in one module, an event is raised and code in another module should be executed, showing additional information
2. Second Iteration: Extended structure. The code executed will depend on the item, raising the event. Executed code does not necessarily show output on the page.
3. Third Iteration: Advanced structure. Events triggering code execution can be configured in the site's administration area. Events and executable code are registered on module installation and unregistered on module uninstallation.

9 XOOPS Core Overview

Before being able to extend the system, it is important to have some background knowledge on the XOOPS core structure.

9.1 Model-View-Controller in XOOPS

XOOPS applies the Model-View-Controller (MVC) design pattern through the use of a mainly object oriented Model and Smarty templates as illustrated in figure 1.

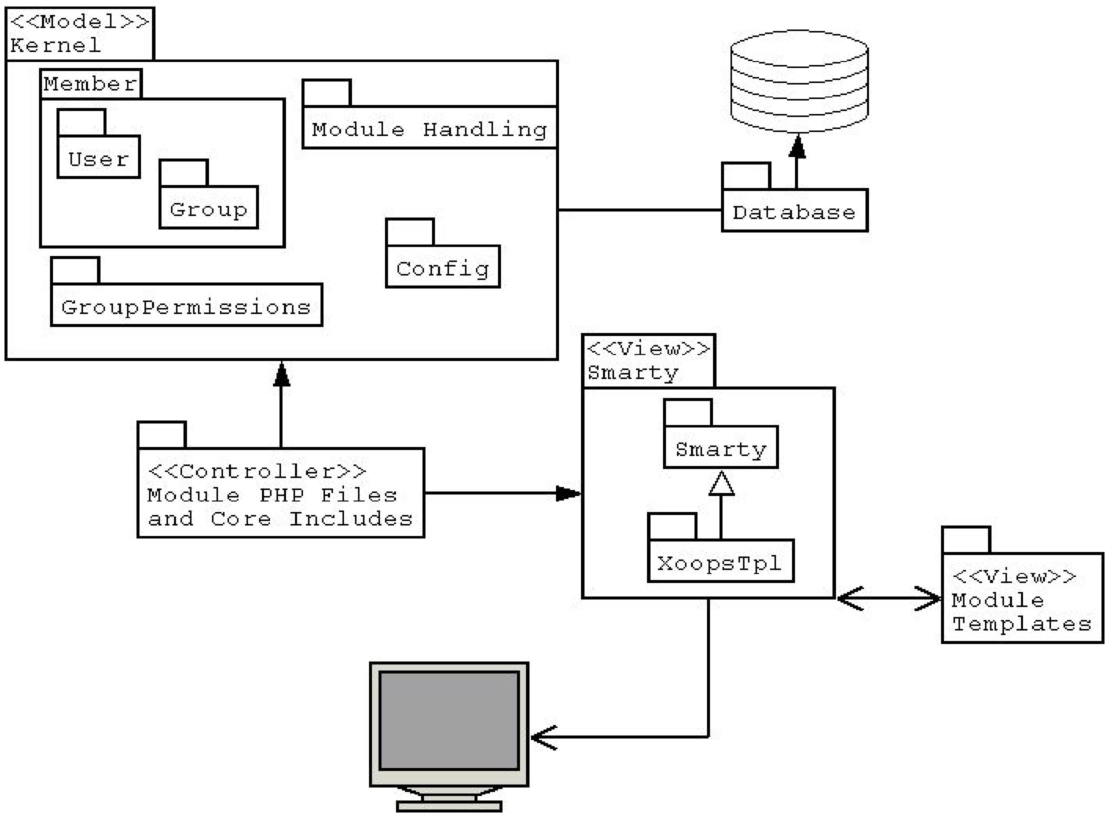


Figure 1: XOOPS Packages and interaction

Module PHP files are accessed by the user through the browser's address field. These files must include a XOOPS core file, which will

1. Include the XOOPS API functions and classes
2. Instantiate the Database object, setting up the connection to the database
3. Instantiate the user object and retrieve user data if the user is logged in
4. Instantiate the current module's object, containing information about the current module
5. Retrieve the current module's configuration options from the database and save it in a globally available array variable

6. Include relevant language files for localized language.

The next step is to include the root file header.php which will create an instance of XoopsTpl, a XOOPS-specific child class to the Smarty class supplied by a third party

With the main parts of XOOPS included, the module file can specify the template to be used for the main content and use module or core classes and functions to retrieve and process data to be shown on the page and “assign” variables to the XoopsTpl object with the assign() method. Variables “assigned” to this object can subsequently be used in the templates. E.g. $xoopsTpl->assign ('varname', 'This is some text'); will make the variable $varname available in templates with the value 'This is some text'.

The file ends with including the root file footer.php, which will use the fetch() method on the XoopsTpl object to read the specified template and assign it to a variable to be used in the theme. The last action is using the display() method on the XoopsTpl object to read the selected theme's main Smarty template, deciding the general look of the page with logos, banners etc. and going through the variables assigned to XoopsTpl holding the blocks and main content, placing them on the page. When the page has been constructed, it is sent to the output buffer and passed on to the browser.

9.2 Main Core Classes

Class programmers can choose to use two core classes; XoopsObject and XoopsObjectHandler

XoopsObject

XoopsObject is an abstract class with a range of useful methods, mainly for property handling, that can be used in a child class directly to ease the setting, getting and sanitation (filtering out harmful elements such as JavaScript code before inserting into the database or before displaying on a page) of properties.

XoopsObjectHandler

Instead of instantiating objects directly, one can use a handler class that extends the XoopsObjectHandler. Objects can then be instantiated with methods on a singleton instance of this handler class.

The main advantages of extending XoopsObjectHandler are:

- A place to put functions working with more than one object (e.g. a "getAllObjects()" function)
- Easier tracking of these functions in the file system. Since they are connected to a class, it is just a matter of finding the class and not searching for functions, which could be in any file in the module or core. This is common for all object oriented code, though, and not specific to XOOPS
- There is no need to include additional files in the page or block code itself as a core function for retrieving handler objects will take care of it
- Avoids retrieval of already retrieved data in the database as these data can be placed in properties on the objectHandler object and used again later

10 First Iteration: Basic Structure

The output of the first iteration is a module that displays a page and allows for hook code in another module to be executed, showing additional information.

10.1 Analysis

First iteration holds one scenario: A page is displayed and subsequently code in another module should be executed, as illustrated in figure 2

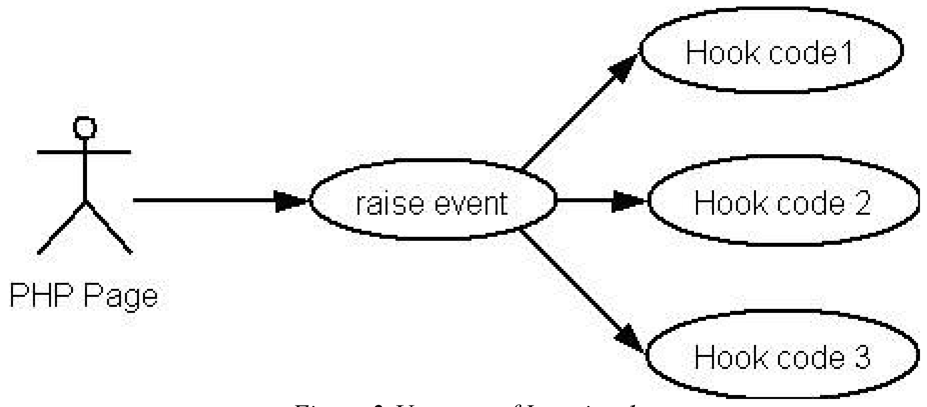


Figure 2:Use case of Iteration 1

I will be working in a new area of XOOPS and in order to separate the code from other parts of the XOOPS Model, it is placed in an individual subpackage. PHP does not have namespaces for classes as is known from other object oriented programming languages so it is not technically possible to distinguish between packages and subpackages in the code and hence this will only be for easier understanding of the solution.

Looking at the use case, we have a need to raise an event and perform an action. Therefore it is logical to create classes for each of these two items; event and action. Following a XOOPS naming convention, the class name will be prefixed with “Xoops” and to further signify that these two classes are in the hook subpackage, I will add “Hook” to the prefix, resulting in two classes; XoopsHookEvent and XoopsHookAction. XoopsHookEvent will be the object of the raised event, whereas XoopsHookAction will be the class executing the hooked code. One event can cause multiple actions to be performed and therefore there is a need to handle multiple XoopsHookAction objects. For this purpose a XoopsHookActionHandler class is created. In later iterations, it will also be necessary to handle multiple XoopsHookEvent objects, so including a XoopsHookEventHandler class from the beginning may save restructuring later.

These handler classes are meant to handle only their "own" classes, so another class will be needed to do the integration between events and actions. A XoopsHookLink class with a matching XoopsHookLinkHandler class seem at present as a good, meaningful name for this link between actions and events.

It should be easy to interact with the hook system, so creating a class as the main interaction class for the subpackage will probably facilitate this. The class should have an easy to remember name and XoopsHookHandler comes to mind as a good, meaningful name.

10.1.1 Attributes and Operations

With the classes identified, necessary attributes and operations can be added to the classes.

XoopsHookEvent

The event needs to be uniquely identified and for that purpose, a unique unsigned integer hookeventid should be assigned to it. This id will not be known when writing the module, but assigned by the system or database, so another way to uniquely identify an event is needed. A string event name could take care of this. However, a name may be duplicated throughout several modules, so it should be connected to the module's id as well. The module id can be retrieved by a method on the XoopsModule class.

The XoopsHookEvent class will in this iteration only need a constructor operation.

XoopsHookAction

Similar to the XoopsHookEvent class, the XoopsHookAction also needs to be identified uniquely in the system, so a similar id, name and moduleid should be attributes in this class.
Since the XoopsHookAction "name" attribute uniquely identifies an action within a module, it can also be used to define which code should be executed, in connection with a naming convention that will be decided later.

Methods on the XoopsHookAction class should display additional information, so to determine the layout of this information, the XoopsHookAction class must contain an attribute, specifying a Smarty template to be used when displaying this information.

The general idea is that modules with hook actions contain classes that extend XoopsHookAction with appropriate operations to perform the actions and specify the template to use.

Apart from the constructor, initialising the attributes, this class should have an operation for calling the operation for executing the hook code and afterwards process the specified template with the use of the XoopsTpl object and assign the content to a template variable.

XoopsHookLink

This class connects XoopsHookEvent and XoopsHookAction objects and will need only a unique ID, the ID of the XoopsHookAction object and the ID of the XoopsHookEvent object.

XoopsHookEventHandler

This class should have an operation for retrieving a XoopsHookEvent objects based on event name and module id.

XoopsHookActionHandler

This class should have an operation for retrieving multiple XoopsHookAction objects

XoopsHookLinkHandler

The XoopsHookLinkHandler should contain an operation for retrieving action ids connected to an event id.

XoopsHookHandler

The XoopsHookHandler will have three attributes in this iteration; the instance of the XoopsHookActionHandler class, the instance of the XoopsHookEventHandler class and the instance of the XoopsHookLinkHandler.

This class combines the other three handler classes and will need an operation with enough parameters to retrieve the raised XoopsHookEvent via the XoopsHookEventHandler instance and retrieve the related XoopsHookAction objects via the XoopsHookLinkHandler and XoopsHookActionHandler instances and subsequently call the hook actions on each XoopsHookAction object.

10.1.2 Class Hierarchy

I have chosen to extend XoopsObject and XoopsObjectHandler in the design of the solution. By doing so, I will have instant access through inheritance to methods for trivial tasks such as setting, getting and cleaning variables on the object classes and an easy way to instantiate the singleton object of the handler classes without actively including needed files – this will be explained in more detail in 10.3.1 : Approach.

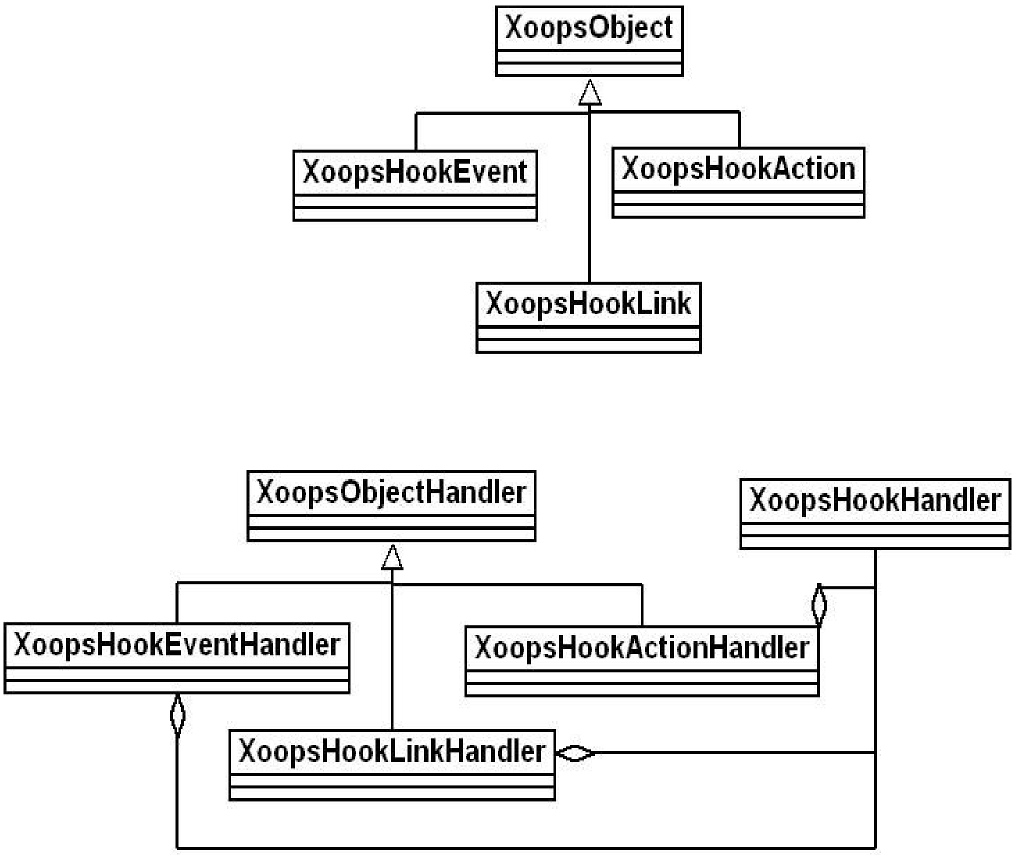


Figure 3: Hook classes overview

As figure 3 shows, the three object classes, XoopsHookEvent, XoopsHookAction and XoopsHookLink extend XoopsObject and their corresponding handler classes extend XoopsObjectHandler. The primary interaction class, XoopsHookHandler does not extend XoopsObjectHandler, as it is not necessary because all actions will be performed on the three other handler classes and the XoopsHookHandler merely joins them together as properties in one class. This approach is also present in the core with XoopsMemberHandler joining XoopsUserHandler (users), XoopsGroupHandler (usergroups) and XoopsMembershipHandler (the link between groups and users)

10.2 Design

10.2.1 Hook output display

Hook code should be a natural part of a page, so the decision on where to place the hook code output should be left up to the module writer.

The XOOPS code for displaying blocks assigns the content of blocks to the XoopsTpl object. A similar approach will be attempted with the hook code.

10.2.2 Database Structure

Actions, events and their linking should be saved in the database as persistent data as these data cannot be saved in the webserver's memory in PHP.

The structure of the data as illustrated in figure 4 follows the object structure with the hook_action and hook_event tables holding information about actions and events, respectively, and the hook_link table holding the relational data between the many-to-many relationship between actions and events.

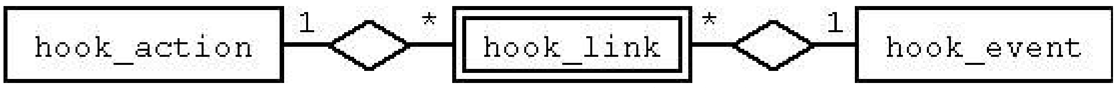


Figure 4: Database relations of the hook area

10.3 Implementation

For the purpose of visualising and testing the hook functionality, two very small modules are created. One for displaying a piece of text and one for applying ratings to items via hooks. The text module has one page, index.php, displaying a certain text from the database decided by ID and raising one event “DisplayText”. The ratings module has no pages to access, but contains a hook for displaying additional content. In second iteration, this content will depend on the text item displayed.

In this iteration, configuration and addition of hooks is simulated by manually inserting data into the database. Since the scope is focused on the core, the modules themselves will also have data inserted manually in the database.

10.3.1 Approach

The functional steps of the hook system can be summarised as follows:

First the XoopsHookHandler class is instantiated with a XOOPS core function

```
$class_handler =& xoops_gethandler('hook');
```

The xoops_gethandler() function will include the necessary files and keep the instance loaded for later use.

Secondly, the triggerEvent() method on the XoopsHookHandler instance is called. This method will use the XoopsHookEventHandler instance to retrieve the object of the triggered event and the XoopsHookLinkHandler to retrieve all action objects, connected to the event. The XoopsHookLinkHandler method *could* return the action ids so the XoopsHookHandler could use the XoopsHookActionHandler to retrieve the action objects, but in order to save a database query, the link and action tables are joined in one single query in the

XoopsHookLinkHandler::getActionsByEvent() method. The getActionsByEvent() method will return an array of objects, each an instance of the XoopsHookAction child class, which must be called DirnameHookAction where Dirname is the directory name of the module containing the action.
Since each of these instances are of a XoopsHookAction child class, they will inherit the callHook() method from their parent class. callHook() will call a method with the same name as the action.

A graphical representation of the approach is shown in figure 5 and two main aspects of the implementation are highlighted in Appendix B.

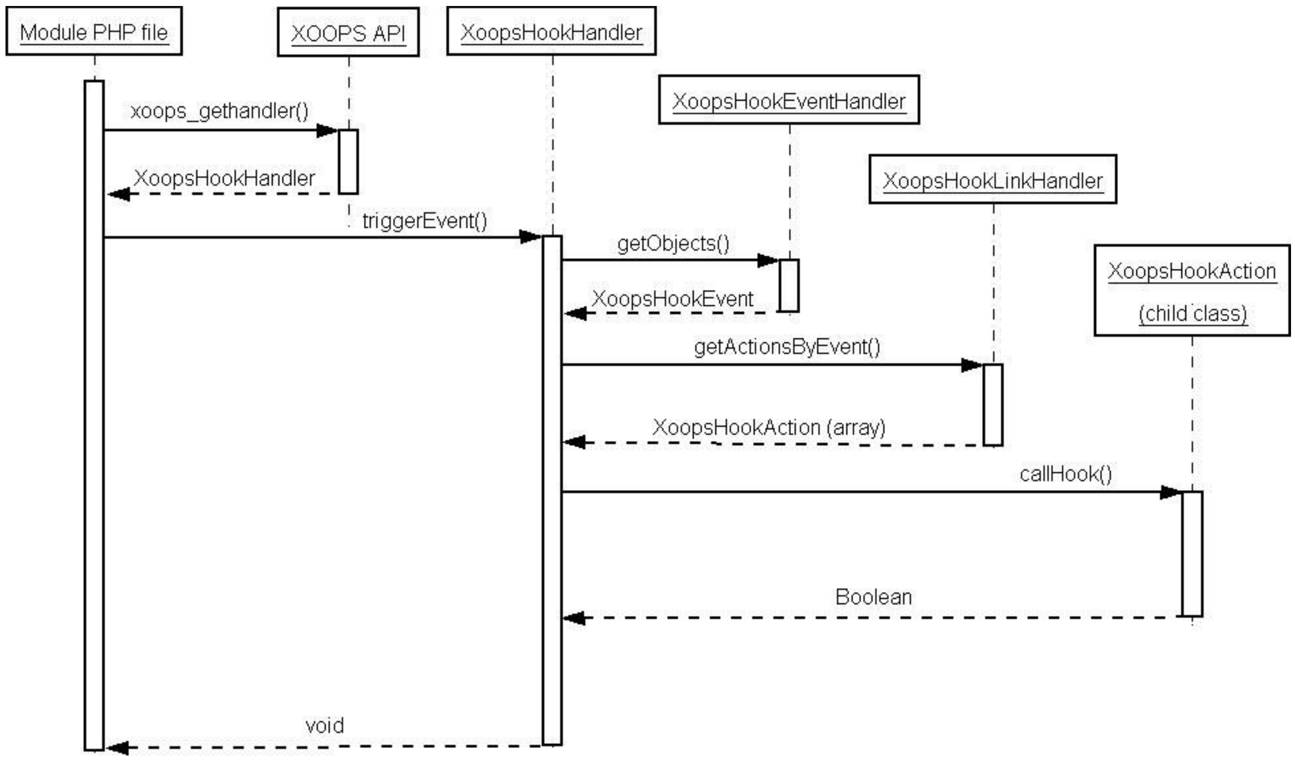


Figure 5: The workings of the hook implementation

10.4 Test

Testing in this iteration consisted of creating example modules for raising an event and responding with an action.

The “text” module is a simple module, displaying a piece of text on the page and raises an event on page display. The “ratings” module only has an action for displaying all ratings in the module's table in the database.

An example text item was manually inserted into the database to be displayed in the module and the “DisplayText” event, which is also manually inserted into the hook tables, is raised on text item display. The “DisplayText” event is linked to the “rating” action through manual database insertion as well. Several example rating records are manually added to the ratings module.

When the text module displays the text an item, the hook functionality should add a new line for each record in the ratings table in the database, saying “The rating is: [rating from database]”. For extra testing, the page was displayed with XOOPS debug settings turned on; in turn PHP debug, MySQL debug and Smarty debug. When displaying the page, it showed correctly and hence, the test passed.

11 Second Iteration: Extended Structure

The second iteration will add the possibility of action code to take an item ID into consideration. Hook code should also take into account that not all actions display information on the page, but could be performing actions such as deleting all ratings for an item when that item is deleted.

11.1 Analysis

Achieving the objectives requires at least two problems to be solved:

1. How can an item in a module be uniquely identified?
2. How should an action without templates/output be executed?

11.1.1 Attributes and Operations

To deal with problem one, an item ID can be passed as a parameter for the triggerEvent method on the XoopsHookHandler class. This ID will also be passed as a parameter for the callHook method on the XoopsHookAction class and passed on to the hook method in the hook module. This will allow the hook method to uniquely identify the item in the module by connecting the event name and module id (which is what the event_id on the XoopsHookEvent class does) with the item id.

Problem two requires a new attribute on the XoopsHookAction class to determine whether or not the hook method has output. An “action_type” attribute can specify that, for example by setting the attribute to “display”. This will also open up for grouping actions together by function such as “add”, “update” and “delete”, although this is of little concern in this iteration.

11.2 Design

Adding a type attribute will require that the module hook method called has an additional element in the name, corresponding to the type. By that logic, the “rating” method on the RatingsHookAction will need to be renamed to “rating_display” and the method implementation will need to identify an item on eventid and itemid combined.

11.3 Implementation

The code in second iteration is very similar to the first iteration except for the added properties mentioned in 11.1.1

11.4 Test

Testing in this iteration consisted of testing whether the added information on text display is only the element, connected to the text item

The “text” module from the first iteration was not modified. The “ratings” module's action method was renamed to rating_display and a second parameter, item_id, added. The item_id is used in the method to only retrieve the records corresponding to the event and item.

The ratings database table was changed to include the item_id and one record was given the item_id corresponding to the text item in the “text” module, whereas the other record was given a value for an item_id that did not correspond.

When the text module displays the text an item, the hook functionality adds a new line for each record in the ratings table in the database that corresponds to the text item, saying “The rating is: [rating from database]”. This was also the observed behaviour and the test passed.

12 Third Iteration: Advanced Structure

The purpose of the third iteration is to remove the manual connection between actions and events. Events triggering hook execution can be configured in the site's administration area after being created on module installation – and will be deleted on module uninstallation.

12.1 Analysis

The advanced structure should answer these questions:

1. How does the system "know" which actions and events a module contains?
2. How will the actions and events be inserted into the database?
3. How should actions be connected to events?

12.1.1 Use Cases

The main use cases in this final iteration is the registration, un-registration and connection of actions and events as illustrated in figure 6

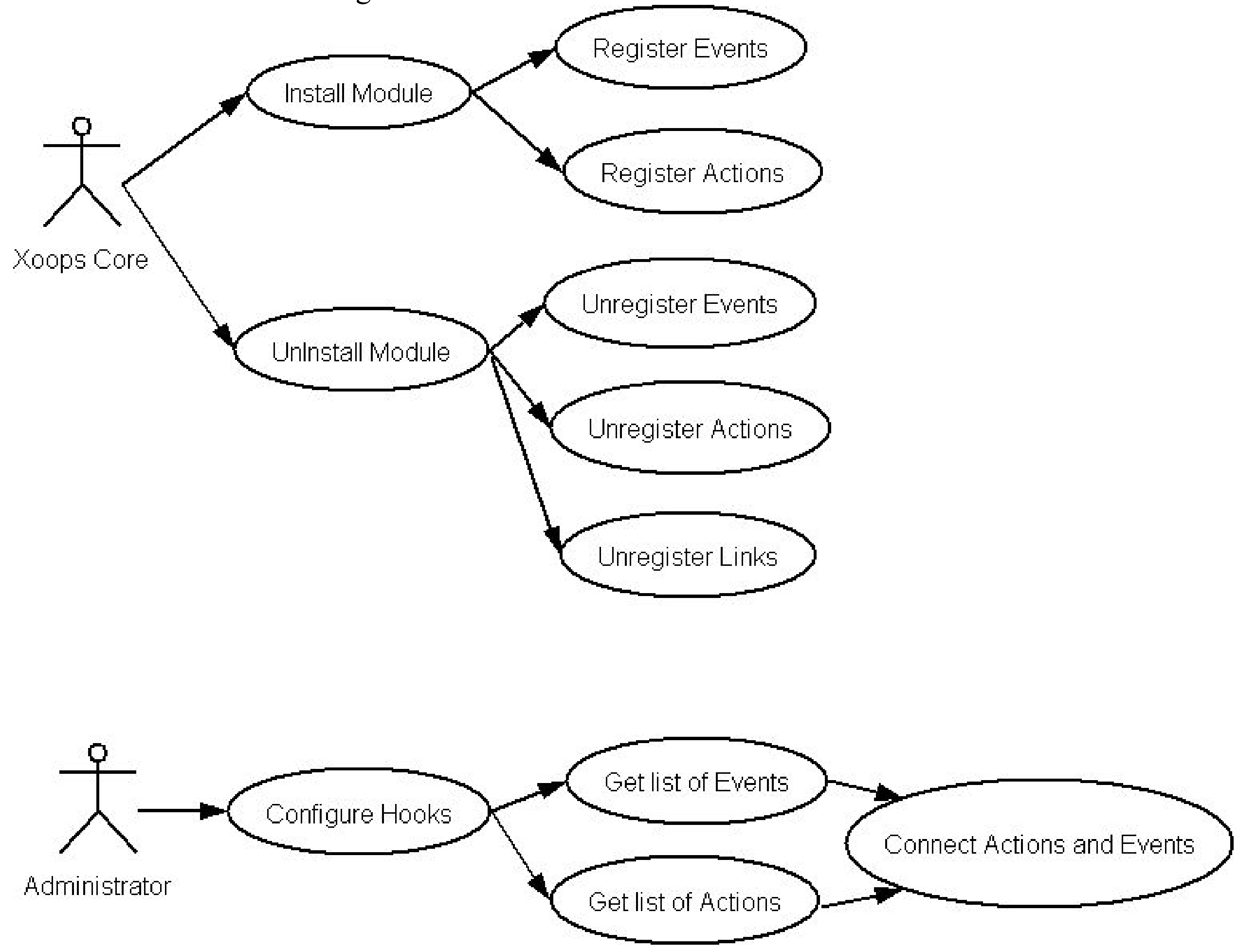


Figure 6: Use Cases for iteration 3

Register Events

In order to register an event, the system must first know that it is there. A module's xoops_version.php file in the module root directory is already used for specifying the module's menu items and configuration preferences as well as configuring the built-in notification and comment systems. Adding hook configuration to that file seems an obvious choice.
An event is identified by either an id or by a name and module id combined. When dealing with the xoops_version.php file, the module id is already known by the system, so only the name needs to be specified in xoops_version.php. An operation is needed to actually inserting the event into the database.

Register Actions

Similar to event registration, actions can be specified in a module's xoops_version.php file. An action is also identified by module id and name combined with the action type, so what needs to be specified in xoops_version.php is action name and type. As with events, an operation is needed to insert the action into the database.

Unregister Events, Actions and Links

When a module is uninstalled, all the module's events, actions and links must be removed, for which one or more operations are needed.

Configure Hooks

Hook configuration is the only aspect of iteration 3 that involves displaying information. The information should be a list of events and a list of actions. When clicking on an event, it should be bring up a page with the all actions, sorted by whether they are linked to the event or not. From that page, it should be possible to link or un-link actions from that event.
Similarly with actions, clicking on one should bring up a page showing all events, again sorted by whether they are linked to the action or not.
For this to work, operations are needed to perform the following:

1. Get all events
2. Get all actions
3. Get all links
4. Save link in database
5. Delete link in database

Interaction with XOOPS kernel

For the registration and un-registration of events and actions, it is necessary to add code to the

modules administration section to call the hook package and perform the needed operations.
It will also be necessary to add new administration code for configuring the hooks. Since it is rather manual work, the process of implementing this will only be described, where it involves interaction with the hook package.

12.2 Design

Following the analysis, some classes will need additional operations, no additional attributes are needed.

For ease of use, the XoopsHookHandler class should be the only interaction class of the handler classes and will therefore need operations for creating actions, events and links as well as insertion into and deletion from the database.

12.3 Implementation

12.4 Test

Testing in this iteration consisted of

- installing the text and ratings modules with hook actions and events
- connecting the DisplayText event in the text module and the rating action in the ratings module through the administration area.
- Uninstalling the text and ratings modules

12.4.1 Test cases

This iteration has more tests than the other two iterations:

1. Hook recognition and database insertion on module installation
2. Removal of hook elements on module uninstallation
3. Connection of actions and events through the administration area

Test 1 – Hook recognition on module installation

The text and ratings modules are installed through the XOOPS administration interface

During installation of the ratings module, the following message should be displayed:

Hook action **rating** inserted successfully

During installation of the text module, the following message should be displayed:

Hook event **DisplayText** inserted successfully

And the rating action and DisplayText event should show up in the hook configuration area of the system administration. This was also the case when tested and the test passed.

Test 2 – Hook Removal on Uninstallation

The text and ratings modules are un-installed through the XOOPS administration interface
During un-installation of the ratings module, the following message should be displayed:

Hook action **rating** deleted successfully

During un-installation of the text module, the following message should be displayed:

Hook event **DisplayText** deleted successfully

And the rating action and DisplayText event should no longer show up in the hook configuration area of the system administration. No errors were encountered, test passed.

Test 3 – Connecting Actions and Events

The hook tables in the database are cleared of all data before the ratings and text modules are re-installed. In the hook configuration area, the DisplayText event is selected and a page showing all available actions in a form is displayed. A checkbox next to the rating action is checked and the form is submitted.
A test item (“This is a test item”) is inserted into the text module's database table and a record (rating = 5) is manually inserted into the ratings module's database table, connected to the DisplayText event and the test item as in the previous iterations.

After submitting the form in the hook configuration area and inserting the records into the database, displaying the test item should show:

This is a test item

The rating is: 5

Which was also observed when testing. After un-connecting the hooks in the hook configuration area, the rating message was no longer displayed. Test passed.

13 Discussion

When performing a comparison between products, questions always arise. Are the compared products representative for the product group? Are the selected products the most suitable products to compare with?
The answers to these questions are that with the rising number of CMS's available, it is virtually impossible to make a representative comparison. I have tried to select the best available, but since this is a subjective evaluation, there will always be people, who disagree.

There is also the question of choosing the area of XOOPS to improve. Was it the most needed improvement? Is the implementation good enough? Is it finished?
I personally think that a hook functionality in XOOPS will enable a wide variety of enhancements in modules and the core alike. During the process of implementing a solution, I imagined several enhancements possible with the hook system; modules adding information to the user's profile, forum threads being created on news article approval and related links between articles, downloads and forum threads.
The hook functionality as implemented in the three iterations is usable, but it is not finished yet. Before it is a viable solution and usable, the following elements need to be in place:

1. Actions should be able to receive additional information about the item that raises an event and use this in the action code – such as the “create forum thread” action, in which it would be useful to have access to the text and title from the event-raising item, so the thread could start with the item text
2. Modules will need to be able to specify actions for selected events only – it does not make sense to have a link between an action and an event if the action is only supposed to be executed on module installation or similar.
3. Modules should be able to specify default links, so the webmaster will not need to configure everything himself on module installation
4. Actions and events should have additional properties, explaining its use for easier configuration
5. Hook registration and unregistration is only performed on module installation and uninstallation, not on module update
6. The XOOPS core should be configured to raise events and hold actions, which modules can take advantage of

With these six elements, I believe that the hook functionality in XOOPS can enhance the flexibility and possibilities in XOOPS immensely.

14 Conclusion

In the process of this project, I have looked at many CMS solutions, ending up with a manageable number of solutions that were subsequently compared to the XOOPS system.

The study showed that XOOPS is a capable system with good flexibility, but also a system, which has existed for a while with legacy code and backwards compatibility interfering with the easy understanding of the features available. Also, the XOOPS module architecture does not facilitate the interaction between modules. This is the main reason why a hook extension for XOOPS, allowing modules to interact through raising events and other modules picking up on these events and performing additional actions, was chosen as the improvement with the highest potential for improving XOOPS as a whole.

The implemented code allows for modules to raise and respond to events and is configurable in the administration area, however the solution is not ready for production until additional functionality is added to the hook package.

Appendix A: Terminology

Some abbreviations and concepts explained:

CMS	Content Management System. System to manage content by providing a framework usually including user authentication and general navigation
Module	Self contained collection of code performing a specific function. Examples could be a News module for announcements and a forum module for discussions
Block	Small, summary-like portion of code that can be placed on pages of individual modules and give limited interaction. This could be a list of the 5 most recent postings in the forum or best rated articles
Theme	“Skin” of the website, a theme controls the general layout (positioning and colouring) of blocks, banners and the area with module output
Template	Individual templates define the area of e.g. blocks and modules where the content goes – as opposed to the theme, which only controls the outlying area – using HTML tags to control the structure, with the actual contents being supplied dynamically. Several technologies exist for using an external framework for themes and templates.
Hack	Modification of code in order to improve or change the behaviour. Usually performed by someone else than the original creator. Quite often hacks are accepted by the original creator and implemented in later versions, so a hack in this definition is not a malicious piece of code.
Fork	When a project is split up in different projects, one is said to be a fork of the other. Usually this happens, when the core developers reach a disagreement and cannot work together – possibly because they want to go in different directions with the architecture – and split up in two or more projects, each following their separate directions. A fork can be a painful

experience for a CMS, suddenly dividing the developers as well as the community, but it can also be a breath of fresh air, when new ideas are being brought in, possibly in a very thorough rewrite of the system.

Appendix B: Important aspects of Iteration 1

An important part of the functionality is how to instantiate the correct XoopsHookAction child class. This is done in the getActionsByEvent() method on the XoopsHookLinkHandler class, with the following code:

```
//Get instance of XoopsModuleHandler
 $module_handler =& xoops_gethandler('module');
  while ($myrow = $this->db->fetchArray($result)) {
    //Retrieve XoopsModule object for this action's module
    $thismodule =& $module_handler->get($myrow['action_moduleid']);
    //Get directory name
    $dirname = $thismodule->getVar('dirname');
    //Include action module's hook class file if it exists
    if (!file_exists(XOOPS_ROOT_PATH."/modules/$dirname/class/hook.php")) {
       return false;
    }
    include_once (XOOPS_ROOT_PATH."/modules/$dirname/class/hook.php");
    //Build name for action class, based on module directory name
    $className = $dirname."HookAction";
    //Instantiate class
    $thisaction =& new $className();
    //Assign values to attributes
    $thisaction->assignVars($myrow);
    //Add this object to the return variable
    $ret[$myrow['action_id']] =& $thisaction;
    //Free objects to be used in the loop
    unset($thisaction);
    unset($thismodule);
  }
```

First the XoopsMemberHandler instance is retrieved before going through each row of the preceding SQL query (not shown). In each iteration of the while loop, the action's module object is retrieved and the directory name extracted. The directory name is used to first include the class/hook.php file from the module directory and subsequently instantiating the “dirnameHookAction” class, which will uniquely identify a class in the entire system.

The most complicated part of the functionality at this time is how to arrange the displaying of hooks, but luckily there is a similar functionality already present in the core: The displaying of blocks.

The code for calling the hook action method is located on the XoopsHookAction class and looks as follows:

```
function callHook($event) {
    //Access the global Smarty object
    global $xoopsTpl;
    //Build method name
    $method_name = $this->getVar('action_name');
    //Call hook action method
    $result =& $this->$method_name($event);
    if (!$result) {
        return false;
    }
    //assign the result of the method to the Smarty object
    $xoopsTpl->assign_by_ref('hook', $result);
    //fetch and process the hook template
    $xoopsTpl->append('hooks', $xoopsTpl->fetch('db:'.$this->getVar('action_template')));
    //clear the assigning so it can be used again in other hooks
    $xoopsTpl->clear_assign('hook');
    //return successful
    return true;
}
```

The main elements of the method are the building of the method name and the fetching of the template. The method name is in this iteration the action name. The template is processed by assigning the result of the hook method to a Smarty variable, usable in the fetch() method on the Smarty object.