An Architecture to Integrate Content Management Systems and Web Applications

Leonelo Dell Anhol Almeida
Vagner Figuerêdo de Santana
Diogo Moreira Bispo
Maria Cecília Calani Baranauskas

Technical Report - IC-08-08 - Relatório Técnico

April - 2008 - Abril

An Architecture to Integrate Content Management Systems and Web Applications

Leonelo Dell Anhol Almeida[*] Vagner Figuerêdo de Santana[†]
Diogo Moreira Bispo[‡] Maria Cecília Calani Baranauskas [§]

Abstract

Some Content Management Systems (CMS) offer a number of facilities to make available different services through the Internet; however these facilities may not meet all requirements of a specific project. In addition, when developing an extension or plug-in that offers a specific service for a CMS, the developed software becomes highly coupled with the platform used. To address this problem we propose an architecture that integrates CMSs with Web applications so that they can evolve independently. From the proposed architecture it is possible to develop services for CMSs to come in the future.

1 Introduction

Content Management Systems (CMS) aim to support the content management and communication about this content through the Web. Bergstedt et al. (2003) pointed out some basic principles of CMS tools: separation of structure, content, and presentation; workflow management; and content management. The initial proposal of CMS tools focused on the management of large amounts of data, probably for the context of enterprises. However, due to some factors such as: facilitated access to the Internet, increase of information sharing among distributed people, improvement of user interfaces, etc; CMS became a very interesting way to organize, interact and share content with other people.

Robertson (2003) pointed out some of the reasons people use this kind of application:

- independence from web designers;
- changes can be made at any time;
- technical details are resolved by the CMS;

[*]Institute of Computing, University of Campinas, 13081-970 Campinas, SP. Partial research support by FAPESP — Fundação de Amparo à Pesquisa do Estado de São Paulo, grant #2007/02161-0

[†]Institute of Computing, University of Campinas, 13081-970 Campinas, SP. Partial research support by CAPES — Coordenação de Aperfeiçoamento de Pessoal de Nível Superior.

[‡]Institute of Computing, University of Campinas, 13081-970 Campinas, SP. Partial research support by FAPESP — Fundação de Amparo à Pesquisa do Estado de São Paulo.

[§]Institute of Computing, University of Campinas, 13081-970 Campinas, SP.

- control of who can do what;
- consistence provided by using themes.

Leveraged by these new interests, CMS developers added some features to the content management mechanisms. The majority of such features consisted of the addition of communication tools such as forum, blog and instant messaging. Furthermore, a number of CMS development communities raised and, in consequence, a lot of CMS were built. In the CMS Matrix portal[3], which compiles information about CMS tools, there are 886 tools registered. Many of these tools have short life cycles and consequently many users face a lack of support and updates.

Because of these problems, in the scope of e-Cidadania Project[5], we propose a new architecture based on the principle of keeping a wider independence degree among system contents and CMS tools. With this approach we expect to avoid work in specific CMS technologies, by producing middleware to translate independent Web content into the appropriate CMS content.

To validate the proposed architecture we introduce a case study within the scope of the e-Cidadania, a Project [5], which aims at the development of technologies to support inclusive social networks. e-Cidadania has the requirement that the software produced in the project must be available under the Berkeley Software Distribution (BSD) License[1]. Considering that the majority of well accepted CMS are under GNU General Public License (GPL)[2] we believe the architecture can bring important contributions in allowing their use without violating the licenses.

In Section 2 of this Technical Report we present the steps taken during the architecture construction; in Section 3, we present the process conducted to choose CMS for the context of the Project; in Section 4, we present the architecture proposed in this work; finally, in Section 5, we draw some considerations about the architecture and further work.

2 Method

The architecture proposed in this work was guided by project requirements and the need for building working prototypes. Consequently, the goal was initially to reuse an available Open Source and cross platform CMS, reducing the effort needed to develop working prototypes.

We searched for the CMSs used by reliable websites (e.g., Brazilian Government Portal, ACM Portal, Utah State University, The New York Times, GE Asset Intelligence, Spread Firefox) and by the ones that offer services correlated to the project needs (e.g., social network, products exchange). Then, we came up with a list of four CMSs: Drupal [4], Plone [2], SilverStripe [8, 9], and Typo3[10]. Moreover, we tried to reduce the comparison scope so we could test empirically the remaining CMSs.

To analyze the systems, we first searched for a comparison template, but we realized that we had to take into account specific requirements [6]. Then we built a list of requirements and features that we are considering in the project and performed a comparison among

[1]Berkeley Software Distribution (BSD). Available at: `http://www.bsd.org/`.
[2]GNU General Public License (GPL). Available at: `http://www.gnu.org/copyleft/gpl.html`.

the referred systems. After this first analysis, we reduced the scope from 4 to 3 CMSs. Nevertheless, it was not possible yet to clearly select one of the selected 3 CMSs. Then, we consulted the CMS Matrix portal that contains a database of more than 8 hundreds CMSs and an extensive list of characteristics and features for each of them. Even after that exploration, the 3 previously selected CMSs kept the more adequate to our needs, so we proceeded to empirical tests with them. The empirical tests involved the tasks of installation, configuration, installation of extensions, and creation and edition of templates/themes.

Before concluding the empirical tests a problem raised: as soon as we focus on the development of CMS's extensions, they will always depend on the chosen system. Then, it was necessary to study the architecture of CMSs so that we could work on a method to allow us the use CMS's functionalities without turning any Web application that use it into a CMS dependent application. After that, we built an architecture that integrates CMSs with the Web application and, consequently, makes possible to use different CMSs within the same project.

3 Preliminary Results of CMS Analysis

The e-Cidadania Project aims at proposing solutions to interaction design and user interface design of computational systems to support citizenship relations in inclusive social networks. The porspective application will be implemented for a target community in joint actions of the responsible research group and partners from the community. From these characteristics, some specific requirements will be addressed in the application: accessibility, usability, tailorable interfaces, etc.

3.1 CMS Evaluation

Since there is no best list of requirements for comparing CMSs [6], the comparison (see Figure 1) was made based on the following requirements of the e-Cidadania project:

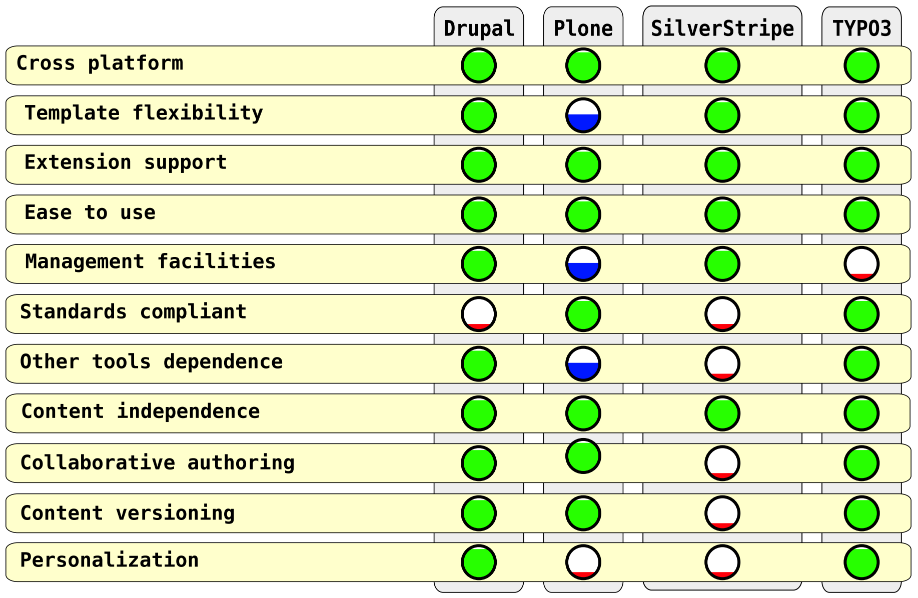
1. **Cross platform** - Measures if the CMS can be hosted in any platform server;
2. **Template flexibility** - Measures how flexible the CMS's templates are so that any change in the user interface can be made from the administration interface;
3. **Extension support** - Measures if the CMS supports extensions or plug-ins with new functionality;
4. **Ease to use** - Measures how easy is to create and maintain content in the CMS;
5. **Management facilities** - Measures how easy is the CMS to be configured;
6. **Standards compliant** - Measure if the CMS is compliant with standards regarding markup language (e.g., HTML, XHTML), style sheet language (e.g., CSS), and guidelines (WCAG);
7. **Other tools dependence** - Measures if the CMS depends on uncommon softwares or platforms configuration;

8. **Content independence** - Measures if the presentation keeps independence of website structure and content. This makes possible the publication of content in different formats (e.g., RSS, WAP);

9. **Collaborative authoring** - Measures if the CMS provides some functionality for collaborative content development;

10. **Content versioning** - Measures if the CMS provides some tool so that administrators and users can control content versions;

11. **Personalization** - Measures how users can change the CMS's user interface to adequate it to their preferences.



	Drupal	Plone	SilverStripe	TYPO3
Cross platform	🟢	🟢	🟢	🟢
Template flexibility	🟢	◒	🟢	🟢
Extension support	🟢	🟢	🟢	🟢
Ease to use	🟢	🟢	🟢	🟢
Management facilities	🟢	◒	🟢	○
Standards compliant	○	🟢	○	🟢
Other tools dependence	🟢	◒	○	🟢
Content independence	🟢	🟢	🟢	🟢
Collaborative authoring	🟢	🟢	○	🟢
Content versioning	🟢	🟢	○	🟢
Personalization	🟢	○	○	🟢

Figure 1: Comparison among CMSs

Figure 1 shows that Drupal and SilverStripe do not guarantee compliance with accessibility standards like WCAG, for example. However, these standards can be achieved in the final applications since they count on flexible templating facilities. In addition, the comparison shows that Plone templating is not flexible at the same level of the other evaluated CMSs, since it needs too much coding (CSS and/or Python) to build a Web page layout that differs from Plone-like websites. Also, Plone management is affected by the dependence of knowledge on how to configure and use Zope application server, affecting evaluation on the dependence of other tools. Typo3 was evaluated as harder to configure since it depends on a specific language (TypoScript), but it was also considered one of the most complete.

SilverStripe is one of the newer CMSs and does not offer the same facilities and extensions than the others. Some of these services are collaborative authoring and content versioning, for example. In addition, SilverStripe depends on specific setup (web server, data base management system, and application server). Moreover, SilverStripe and Plone users don't have many options to configure or personalize the manner they use and see the CMS's interface.

The comparison was based on the characteristics shown by CMSs' providers and resulted in the elimination of 1 of the 4 tools initially chosen. Once the initially selected CMSs were Drupal, Plone, and Typo3, we proceeded the empirical analysis of installation, configuration, administration, and use. Thus, we compared them using the CMS Matrix Portal and found that they have in common the following characteristics (either natively or by module extension):

1. Content versioning
2. Active developers community
3. Online help
4. Support to internationalization
5. Public forums
6. APIs
7. E-mail lists
8. Server Page Language
9. Online administration
10. XHTML compliance
11. Chat
12. Wiki
13. Blog
14. WYSIWYG editor
15. Discussion/Forum
16. Event calendar
17. Search engine
18. Site map

Another source of comparison among the selected CMS was the empirical tests (see Apendix A). These tests evaluated the facilities to install and use the systems and their capacity to add new functionalities, specially the ones to support social networks. From the tests we identified that, regarding the facility to install, all the CMS are quite simple. Otherwise, in the evaluation of the ease of use and configuration, we verified that TypoScript, the Typo3 script language, is not easy to learn. In addition, We identified that Drupal offers more social network extensions than Plone and Typo3, that count on few options.

Since the comparison does not revealed a representative order among the compared CMSs, we agreed that any of the selected CMSs would meet the project requirements. However, the CMS initially chosen is the one with the best results in the empirical tests; this does not mean that it will be the best choice all the time since the requirements may change. We are considering an architecture that also allows the change to other CMS.

4 The Proposed Architecture

The typical architecture of a Web application based on a CMS is shown in Figure 2. The CMS core has the basic rules, functions (e.g., content indexing) and mechanisms (e.g., themes, communication tools). Information as user profiles and metadata are usually stored in external Database Management Systems (DBMS). Robust CMSs offer the capacity to extend the CMS core. Extensions are an important component of CMSs due to the unlimited possibilities of contributions from communities, enterprises and individual users. Systems as Plone, Drupal, and Typo3, release a significant part of their resources as extensions. As presented in the Section 3.1, many of the common resources identified as requirements to the e-Cidadania Project would be provided by extensions.

Users interact with the CMS tools by two basic types of interfaces: administrative interface (e.g., management of themes, configurations) and user interface (i.e., product provided by Web applications). Both CMS core and interfaces run under a Web Server.

Considering the complex scenario faced in providing content independence from CMS without losing functionality provided by these systems, we propose an architecture to integrate CMS and other Web applications with a lower dependence degree (see Figure 3). With this approach we believe that small groups and individuals who maintain Web applications will be able to change to other CMSs without losing work and content.

The main change regarding the traditional architecture is the separation of the Web application core from the CMS core. With this separation, we can still use the functionalities of the CMS without creating a strong dependence on it. To make this possible it is necessary to provide a communication channel between the cores which is done by using extensions. This approach provides flexibility to a Web application to be used in more than one CMS system at the same time without losing generality and without CMS specific codification.

In the context of the e-Cidadania Project, the use of extensions create conditions to the development of BSD Web applications using a GPL CMS system without violating any license. Another difference is the explicit separation among Web application data and CMS system data. Even when they use the same SGBD, they are not obligated to share the same database instance.

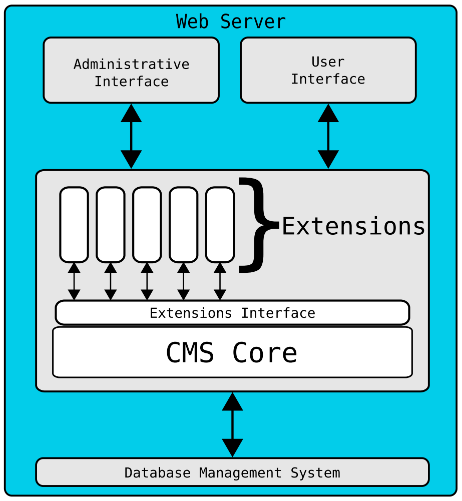


Figure 2: A typical architecture of a Web Application based on a CMS.

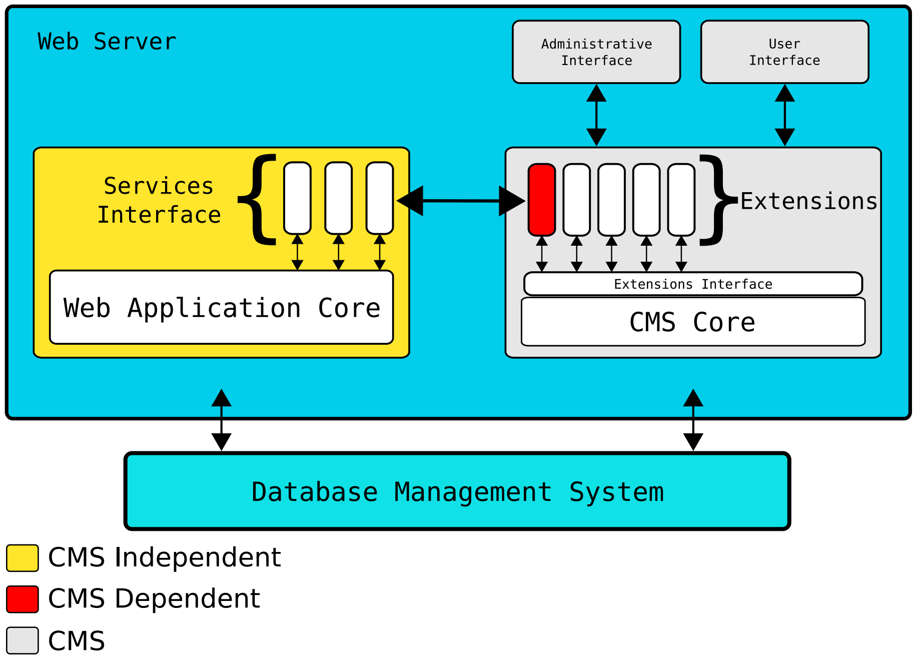


Figure 3: An architecture to integrate CMS and Web applications with a low dependence degree.

The extension (filled in red in Figure 3) is a CMS specific sotware, which translates CMS's requests to an standardized common XML markup for Web applications. On the other side, there is a set of extensions responsible to read the requests made through the CMS and to translate it to Web application specific code. In this way, it is possible to develop services in a completely independent way of the CMS.

Information such as the user registration can be stored both in the CMS's database (e.g., proceed the authentication) and in the Web application's database (e.g., to profile). Therefore, considering that the architecture should provide support to more than one CMS at the same time, it is necessary to define a structure capable to handle information from different sources presenting them uniformly and keeping track of the sources. To solve possible conflicts due to this situation we use a simple approach from the Database area, in the context of Data Replication. For each information stored in the Web application database coming from CMS we store the location (e.g., IP, CMS instance name, a extension instance signature) of the origin of the information.

With the proposed architecture we create conditions to the development of Web applications which explore some resources from CMSs without becoming dependent on them. The services interfaces provide a common ground to communicating with different tools with low additional effort and, consequently we avoid possible license restrictions due to the integration among these tools.

5 Conclusion

Due to the variety of CMSs it is usually hard to define a better system for a specific project. Based on the requirements elicitation, we could select one and reuse its functionalities in a Web application. However, the initial requirements may change during the project design and the application would be CMS dependent. To cope with this problem, we proposed an architecture to integrate Web applications with different CMSs without making the applications dependent on the technology in use.

We found that the usual architecture of Web applications based on CMSs focuses on the development of extensions. However, these extensions depend on the CMS and, even, on a specific CMS's version. So, the proposed architecture foresees the communication among Web applications and CMSs in a way that they can evolve independently. The proposed architecture makes possible that Web application developers reuse CMS's functionalities and focus on the specific services they are going to build.

Finally, this work may help developers to integrate their Web applications in future concept proofs, since CMS's new versions and CMS migration may occur.

Next steps in this work include the implementation of the extensions proposed in this work and the validation of them in the context of the e-Cidadania Project, initially using Drupal as CMS. Some interesting future research themes are: performance evaluation of the proposed communication model, requirement elicitation and implementation of extensions to other CMS systems and the facility to develop service interfaces.

Acknowledgments

This work is funded by Microsoft Research - FAPESP Institute for IT Research (grant #2007/54564-1). The authors also thank colleagues from NIED, InterHAD, Casa Brasil, CenPRA and IC Unicamp for insightful discussion.

References

[1] Bergstedt, S., Wiegreffe, S., Wittmann, J., Moller, D. (2003). "Content management systems and e-learning systems -a symbiosis?", Advanced Learning Technologies, 2003. Proceedings. The 3rd IEEE International Conference on (9-11 de Julho, 2003), pp. 155-159.

[2] Burton, J. (2007). "What is Plone?"Available at: `http://plone.org/about/plone`. Last access: April, 09, 2008.

[3] CMS Matrix. Available at: `http://cmsmatrix.org/`. Last access: April, 09, 2008.

[4] Drupal - Features. Available at: `http://drupal.org/features`. Last access: April, 09, 2008.

[5] E-Cidadania Project Portal. Available at: `http://styx.nied.unicamp.br:8080/ecidadania/`. Last Access: April, 09, 2008.

[6] Robertson, J. (2002) "How to evaluate a content management system", Step Two Design - K M Column. Available at: `http://www.steptwo.com.au/papers/kmc_evaluate/index.html`.

[7] Robertson, J. (2003). "Why very small website needs a CMS", Step Two Design - CMS Briefing. Available at: `http://www.steptwo.com.au/papers/cmb_needcms/index.html`. Last access: April, 09, 2008.

[8] SilverStripe. Available at: `http://www.silverstripe.com/`. Last access: April, 09, 2008

[9] SilverStripe Documentation - Server Requirements. Available at: `http://www.silverstripe.com/`. Last access: April, 09, 2008.

[10] Typo3 - System Requirements. Available at: `http://typo3.org/about/system-requirements/`. Last access: April, 09, 2008.

Appendix A

CMS Evaluation - Empiric tests report

Goal

Start the implementation of a social network website using 3 CMSs: Drupal, Plone and Typo3; reporting the differences in each one. To perform the evaluation we selected one of the members of the e-Cidadania project who is an under-graduate student and does not have any preview experience with CMS systems. The initial goal was to follow the steps below, reporting the activities in detail:

1. Install the 3 CMSs;
2. Install/Implement the features selected on February 8, 2008. At least: user's profile, user's contact list and communities and;
3. Customize the 3 CMSs to have the UI defined on February 8, 2008.

1 Installation

All the CMSs were quite easy to install. The link for downloading Plone was in its main page (`http://www.plone.org`). The installer already comes with everything (i.e., Zope, Plone, Python). There are packages for Windows, Mac and Linux, I got the Windows version. The Plone's Installer only asks for an username and a password (for CMS's administrator) and an installation's path. Once installed, I just opened the browser and accessed the http://localhost and the plone's homepage opened, indicating a correct installation. I verified later that the installer also creates a service called "Zope Instance"that controls the Zope server.

Typo3 was found at `http://www.typo3.org` or `http://www.typo3.com`. There is a link at `http://www.typo3.org/download` to get an Installer, with versions for Windows, Linux and Mac. I got the Windows version, which installed Apache, MySQL, PHP and PHP-MyAdmin, all ready to use. After the installation, I went to `http://localhost:8502/` and a Typo3's page opened, indicating a correct installation.

There was no Drupal installer. I downloaded Apache, MySQL, PHP and Drupal's files and setup them following tutorials I have found on Web,I followed the "Getting-Started"of Drupal[3], the "Quickstart"of Typo3[4] and "Plone 3.0 User Manual"of Plone[5] to get started with the basics of them.

2 Drupal

2.1 Installing new Features

I started with Drupal, because it has the Profile feature ready-to-use, I justset it up. To install a new module in Drupal is simple: just go to Administer-> Site building-> Modules, check the desired modules and click on "Save Configuration". I installed the Locale module (for Localization), and the Profile one. After installation, the link "Profile"appeared under "User Management". In this page I could create customized fields for profiles, setting the compulsory ones.

[3]Drupal, Getting-Start. `http://drupal.org/files/getting-started_2.pdf`

[4]Typo3, QuickStart. `http://typo3.org/fileadmin/dl/tutorials/quickstart.pdf`

[5]Plone, User Manual. `http://plone.org/documentation/manual/plone-3-user-manual/referencemanual-all-pages`

The "Localization"link appeared in "Site Configuration". I had to download a Portuguese translation at `http://drupal.org/project/Translations` to install it. After this I set some user configurations. I set the register as public, so anyone could create an account, and set the user permissions so the users could use the new features.

There were missing two features: list of friends and communities. I looked for new modules on Drupal website and I found them organized by category. In the "Community"category I got the "User Relationships"module and in the "Organic Groups (OG)"section I got the "Organic Groups"module, the "View"module was also needed, because there is an Organic Group dependence. I put all the files in the Drupal modules' directory, and they appeared in the modules list to be installed. I checked the new modules and new links appeared in the menu: "Relationships", "Groups"and "Organic Groups".

In the Relationships section it is possible to manage users' relationships. I created a new relationship named "friend", so users could create friend relationships among them, and I set the permission as in the other modules. Then I created a new account named "newuser", and I added "newuser"as a friend of "admin"and this relationship appeared in the profile of both of them.

I needed to create a new kind of content in the Drupal to configure the Organic Groups module, so I created the "Community"one. In the Organic Groups configuration page, I needed to choose the "Community"type as the Organic Group content. Then I set the users permissions to manage groups and I created a new Community.After the configuration of profile, list of friends and communities, I was ready to customize the Drupal's interface.

2.2 Customizing Drupal

Drupal's interface is based on templates (just like the other CMSs). There are lots of templates ready-to-use, and some of them are available in the default installation. A template is basically formed by a template engine (I used the PHPTemplate, which is recommended by Drupal), a page.php (which is used as a standard for all website's pages) and UI resources (e.g., CSS, Javascript).

So, I made a theme called "tema1". I created a page.php, a style.css, got some icons from Internet for the main menu and configured some menus to be similar to the documentation. It is also possible to customize specific pages to differ from the page.php standard, but I was focused in the general UI.

3 Typo3

3.1 Installing new Features

First, I installed the Portuguese extension. There is a section in the Typo3's BackEnd "Ext Manager"to manage extensions. I downloaded and installed this extension through this section. Apparently it is not possible to get and install translations out of this section (other extensions could be installed off-line). I looked in Typo3's repository for extensions similar to the Drupal's, but I didn't find them or I find alpha and beta versions. Typo3's repository is not organized by category. I used the search engine, without success. So I created a page tree in the BackEnd to represent the pages which was created on Drupal:

- Root
    - Home
    - Profile
        - My profile
        - Edit rofile
    - Friends
        - My friends
        - Friends requests
    - Communities
        - My communities
        - All communities
        - Recent updates

Then I began to customize Typo3.

3.2 Customizing Typo3

Customizing Typo3 is a little bit different from customizing Drupal. While the second uses PHP and CSS, the first uses only HTML and CSS (making independent the job of designer and programmer) and a declarative language, the TypoScript (TS).

I followed the tutorial to customize it, the "Modern Template Building"[6] was really useful.

The final page (that the user can interact) is a combination of dynamic content and HTML template, all organized by the TS. To make this possible, I got the Auto-Parser extension on Typo3's website.

TypoScript is not an easy language to learn and use, because it has a lot of properties and a specific syntax. I created a template for Typo3 using (with some changes) the Drupal template files. But I couldn't put the images in the main menu and in the secondary menu.

4 Plone

4.1 Installing new Features

Plone (as Typo3) doesn't have many extensions for social networking. I tried to install some extensions for Plone and customize it a little, but without success.

[6]Typo3, Modern Template Building. `http://typo3.org/documentation/document-library/tutorials/doc_tut_templselect/current/`

5 Conclusion

After working with the three CMSs, I could see that they have different pros and cons. Drupal is the easiest to use: in few hours anyone could learn the basics. Typo3 separates the work of editors, programmers and web designers. And Plone is stable, with Python language and the possibility to use Python scripts. As disadvantages, I could see that Drupal doesn't separate programmers' and web designers' work; Plone is not very simple to install new extensions and has just a few of them; Typo3 also has only a few extensions and uses a proprietary language.