MŰEGYETEM 1782

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Control Engineering and
Information Technology

Multilingual Web Applications with Open Source Systems

Gábor Hojtsy (gabor@hojtsy.hu)
Budapest, May 18, 2007

Consultant:
Péter Hanák (hanak@inf.bme.hu)

MŰEGYETEM 1782

Thesis Assignment

Multilingual Web Applications with Open Source Systems

Tasks:

1. Define the characteristic requirements of multilingual web sites compared to monolingual implementations
2. Demonstrate and classify some of the popular existing open source systems used for multilingual web sites
3. Explain the examined systems' major weaknesses and the possible solutions
4. Design a prototype implementation for one of the examined systems
5. Implement some of the key elements in the system
6. Summarize your findings and explain further development opportunities

Placeholder page for Hungarian version of the thesis assignment not included in this document.

Declaration

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources credited in the list of references.

Gábor Hojtsy

Nyilatkozat

Alulírott *Hojtsy Gábor*, a Budapesti Műszaki és Gazdaságtudományi Egyetem hallgatója kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, és a diplomatervben csak a megadott forrásokat használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hojtsy Gábor

Contents

Abstract

Modern web sites target people across country borders and within countries where multiple languages are spoken. When serving such an international and multilingual community, we need to take into account several factors in order to support these needs and effectively reach our target group.

In my thesis I investigate these special factors that add to common web sites' needs and often require a different approach to backend development. I also explain some of the standards, recommendations, and best practices that should be followed for this type of web site.

Because most present-day web sites are built on an existing framework that allows developers to reuse established solutions and thus save on development costs, my main targets of examination are these frameworks, the so called content management systems. By comparing and contrasting their strengths and weaknesses as they relate to my focus areas, I devise an implementation plan to fulfill the outlined requirements with the Drupal content management system.

Working with a well-known framework means that my results are critiqued and tested by people interested in using them in real life projects, so the solutions I present here should be both practical for web site implementors and usable for site editors. Although I work in many areas on the multilanguage spectrum, focusing on key aspects allows me to deliver solutions and at the same time open the door for later developments.

My results are freely available to every Drupal user and developer, since they are either integrated into the Drupal core system or are downloadable as Drupal extensions. Further, since the software I have developed is open source, web site implementors can easily adapt it to their special multilanguage requirements by looking under the hood.

Kivonat

A modern webhelyeket sokféle nyelvet használó különböző országok, valamint többnyelvű országok lakosai látogatják. Amikor ilyen nemzetközi és több nyelvet beszélő célközönséghez szólunk, sokféle szempontot kell figyelembe vennünk, hogy speciális igényeiket hatékonyan ki tudjuk szolgálni.

A diplomatervemben ezeket az átlagos webhelyek igényeit meghaladó speciális szempontokat vizsgálom meg, amelyek a háttérprogramok kialakításakor a szokásostól eltérő megközelítést igényelnek. Bemutatom a kapcsolódó szabványokat és ajánlásokat valamint követendő gyakorlatokat, melyeket az ilyen típusú webhelyek készítésekor figyelembe kell vennünk.

Mivel napjainkban a legtöbb webhely egy meglévő keretrendszerre épül, amely lehetővé teszi, hogy fejlesztési költséget is megtakarítva már bevált megoldásokat hasznosítsunk újra, a vizsgálatom célpontjai az ilyen keretrendszerek, az ún. tartalomkezelő rendszerek. Először néhány nyílt forráskódú tartalomkezelő rendszer erősségeit és gyengeségeit hasonlítom össze a többnyelvűsítés szempontjából, majd az igények kielégítésére megvalósítási tervet dolgozok ki a Drupal tartalomkezelő rendszerrel.

Egy ismert keretrendszer alkalmazásának az egyik előnye az, hogy a készülő megoldásokat gyakorlott tervezők és fejlesztők tesztelik és bírálják, ami garancia arra, hogy e megoldások haszonosak legyenek a webhelyek fejlesztői számára és jól használhatók legyenek a tartalomszerkesztők szemszögéből is. A többnyelvűsítés, mint látni fogjuk, sokféle kérdést vet fel, közülük néhány kulcsfontosságú szempontra fogok koncentrálni a diplomatervben. Így használható megoldásokat tudok kidolgozni, miközben a lehetséges későbbi fejlesztéseket is figyelembe tudom venni.

Az eredményeim szabadon és ingyenesen elérhetők minden Drupal felhasználó és fejlesztő számára, részben a Drupal alaprendszerbe beépülve, részben kiegészítő modulok formájában. A kifejlesztett szoftverelemek nyílt forráskódúak, új webhelyek kialakításakor igény szerint átszabhatók.

Chapter 1

Introduction

The World Wide Web was an international space from its start with visitors who speak different languages and reside in various countries. Even with only taking multilingual countries into account (like Canada and Belgium), our web presence needs to cater to visitors speaking different languages. If we add international requirements to our task list, we should also consider cultural differences, local customs, time zones, shipping costs, and other issues.

As the user base of web sites and services expands, it becomes natural to provide interface and even content in more languages. Because existing monolingual web site implementations are often complicated to migrate to a multilanguage model, it is important to keep this issue in mind when planning a new project that might involve support for multiple languages.

Fortunately building web sites has become easier in recent years with many content management systems now available that allow "click and type" web site creation. These systems help users create and manage content, and often even a community, online. Open source content management solutions first widely became popular among small businesses and hobbyists, and eventually big companies and institutions like Yahoo, NASA, Lufthansa and Nokia realized the benefits offered by these systems and deployed them.

Most of these systems provide convenient ways to manage web site's architecture and content added by users and editors of the web site. Although these systems are developed by international communities, multilanguage features are not always integrated into them. As noted, these systems are used in situations with widely different needs, from powering simple blogs to major government web sites (as is the case of Brazil [1]). Many of these different use cases share the requirement to support multilanguage features, even if the exact needs in these use cases are different. A single user blog could have content in different languages, while a complex government site requires parts of its web presence in multiple languages at once.

Mature multilanguage support guides web site visitors to the language version of the content they understand. Content authors and translators can be facilitated with an editorial workflow tailored to their special requirements, possibly including support for interaction with external professional translation service providers.

In the second chapter of my thesis, I look at the challenges multilingual web systems face compared to monolingual implementations and then I specify the areas I will look at in-depth in later chapters. In the third chapter, I examine some of the existing open source solutions, namely Joomla, TYPO3, Plone and Drupal, and look at their approaches to multilingual interface and content management. A comparison of these systems follows in the fourth chapter, based on my focus areas, and it highlights the problems with implementation of some of the desired features. The fifth and sixth chapters present a plan to design an improved multilanguage solution based on my research for the Drupal system, as well as a presentation and evaluation of the actual implementation. Finally, I summarize my work and outline future challenges in chapter seven.

Chapter 2

Multilingual Web Site Requirements

2.1 Terminology

Because the terminology is not clearly defined and is used differently in other papers, it is important to define the basic terms. For my thesis I followed the definitions set forth by the World Wide Web Consortium (W3C) Internationalization (I18n) Activity [2].

Multilingual web site A web site available in multiple languages. Several countries (for example Canada and Belgium) have more than one official language, so a multilingual web site is not necessarily an international one.

International web site A web site intended to be used internationally. This type of web site is not necessarily a multilingual one because residents of multiple countries can speak the same language.

A web site can both be multilingual and international, thus serving people in multiple countries with different languages available. Unfortunately language alone is not always enough to consider when presenting information to a web site visitor.

Locale In computing the locale concept refers to a set of rules for presenting information to a user. Locale includes date formatting, currency, the language variety used, order of sorting and so on.

A multilingual web site should ideally support multiple locales, so *multilocale web site* would be a more accurate term, but this is not used in practice, so I will stick with “multilanguage” in my thesis and only refer to locales when the difference is important.

Two essential terms are used when explaining the process of making a web site multilingual or international. Internationalization and localization are these two keywords and are sometimes used interchangeably although they have very different meanings. Richard Ishida has good definitions [3] of these terms.

Internationalization Also known as i18n, internationalization is the *design and development* of a product, application or document content that *enables* easy localization for target groups that vary in culture, region or language (locale).

Localization Also known as L10n, localization is the *adaptation* of a product, application or document content to meet the language, cultural and other requirements of a specific target market (locale).

This leads to the possibly confusing conclusion that if we want to create a multilingual web site, we need to *internationalize* it, since this is the term used to represent adding capabilities to support multiple locales. Localizing a web site "only" means adding specific features or content for a particular locale.

Although the World Wide Web was designed to offer interconnected **web sites** to visitors, more traditional applications also found their way onto the internet, which resulted in **web applications**. There are subtly different descriptions for these two terms. Web sites can be considered collections of interlinked web pages managed together that allow you to *read* their contents. Web applications, on the other hand, are applications accessed through a web server that allow you to *do* something. While there are clear examples of both, most web sites are now a mix of pages with application-like functionality. The multilingual principles discussed in this thesis are equally applicable to traditional web sites and web applications, so these terms are used interchangeably.

The synergy between these two terms is also driven by **Web Content Management Systems** (WCMS or CMS) that offer convenient content management and application functionality in the same package. The focus of my thesis is web based content, so Enterprise Content Management Systems (built to handle other types of content, like word processing documents and spreadsheets created in the enterprise) are out of my scope.

2.2 Web Standards

When building multilingual web sites, we need to first consider technical requirements and possibilities. Web standards (recommendations and specifications) define our communication means between web servers and clients, so these must be examined first. It is important to note that these standards are applicable to single language web sites too, although they are not widely known in English speaking areas of the world because the defaults provided are adequate there, so there is no immediate reason to think of these building blocks.

2.2.1 Internationalized Resource Identifiers

First we need an address to access a web resource. These days users demand web sites in their own languages, including both the interface and the address.

Web addresses are typically expressed using Uniform Resource Identifiers or URIs. The URI syntax, as defined in RFC 3986 [4], restricts addresses to a small number of characters: upper and lower case letters of the English alphabet, European numerals and a small number of symbols. Unfortunately a URI does not allow for non-English characters, which limits its usability internationally. Internationalized Resource Identifiers (IRIs), as specified in RFC 3987 [5], allow for domain names and paths to contain any Unicode character, thus allowing for fully localized web addresses. (Unicode is explained in the next subsection.)

For IRIs to work, the underlying protocol (HTTP, SMTP and so on) should be able to carry the information, the format used (HTML, XML and others) should support Unicode characters, the applications handling these formats should be capable of dealing with them, and the servers hosting the resources addressed should be able to match IRIs to files and other types of resources. Unfortunately IRI supportive web browsers are not yet widely used as of this writing. While the latest Microsoft Internet Explorer Version 7 supports IRIs, this browser is not yet adopted by mainstream users. Microsoft Internet Explorer 6 only supports IRIs with an add-on installed separately. Other browsers have good support for IRIs as W3C test results show [6].

A basic IRI (eg. `http://árvíz.hu/dokumentumok/védekezés.html`) consists of a scheme (`http://` in this case), a domain name (`árvíz.hu`) and a path component with a directory and a file name (`/dokumentumok/védekezés.html`). More complicated IRIs can contain a port number, HTTP GET parameters and a fragment identifier, which are already adequately covered by existing standards, so IRI support evolves around domain names and path values. Internationalized Domain Names in Applications (IDNAs) are mappings of Unicode strings to special US-ASCII strings, which map to IP addresses in the standard domain name system. For example the `árvíz.hu` name maps to the `xn--rvz-dla6d.hu` domain, `xn--` being a prefix to identify the IDNA encoding, `rvz` being the ASCII characters from the domain name and the `-dla6d` suffix encoding the accented characters. Path components of IRIs are handled by hosts serving the resource (a web server in this case).

Current support for IRIs only allow for IDNAs, which build on the existing Top Level Domain Name (TLD) set. Localized TLDs are still tested by the Internet Corporation for Assigned Names and Numbers (ICANN) for compatibility [7] and will only be available when reliable server technology is present and ICANN rules make it possible to get them registered.

Once widespread support is available for IRIs, multilingual web sites can make use of them. It is a natural requirement that different language versions of a web site be made available under local addresses like `http://www.coffee-bean-ltd.com` and `http://www.kávé-bab-kft.hu`, when branding requirements are not going against localizing the site name.

2.2.2 Character Encoding

Once we have an address, the communication protocol needs to support the encoding of characters used by the desired languages.

The World Wide Web is powered by the HTTP protocol, of which the 1.1 version [8] is used widely, as defined in RFC 2616. Section 3.4 of the RFC explains that HTTP shares the notion of "character sets" with the MIME [9] specification. "Character encoding" would be a better term as described by the HTTP specification, but the term "character set" was kept to stay compatible with the MIME standard.

MIME defines a way to represent multipart messages with headers and content in non US-ASCII encodings. This opened the door for different language encodings to be used in email and later on the web, when HTTP adopted this specification. Responses by web servers include a `Content-type` header, which specifies the content type and character encoding used. A typical encoding for Hungarian documents is ISO-8859-2 (also known as Latin-2), which contains proper accented characters for the Hungarian language. This encoding uses one byte for every character but limits the possible characters to only those used in Central Europe. The biggest online media outlets, such as `http://origo.hu/` and `http://mtv.hu/`, use the Latin-2 encoding in Hungary as of this writing.

Like Hungarian, every language has one or more specific character encodings assigned to it, which can be used to deliver content on the web. However this causes the problem that encoding needs to be tracked and taken care of on every page. To deliver pages in different languages on the same web site we need our backend software to support every encoding we will use and utilize the appropriate one when presenting a web page to a user. One of the bigger problems of this approach is that it is not possible to mix characters from different languages on the same page, such as including a Japanese performers native name in an entertainment news article on one of the above mentioned media outlet pages while using the ISO-8859-2 encoding.

The Unicode Standard (first published in 1991) was designed by the Unicode Consortium [10] to overcome the limitation of traditional encodings and to allow multilingual text presentation. It took many years for the standard to become widely used, but it eventually became a requirement in multilingual systems. Unicode provides a unique code point (a number) for each character, and then different character encodings can be used to map these code points to actual bytes for transmission or storage.

There are multiple ways to encode Unicode characters, of which the most popular is UTF-8 encoding because it is the most compatible with existing ASCII systems and still enables users to simultaneously use the full Unicode character set. UTF-8 is a variable-width encoding, using one to four bytes per code point. UTF-8 is now the de facto standard for multilingual web environments, and so this is the encoding I have chosen to use throughout my thesis. This allows for multiple language characters to be used on the same web page as well as allows common algorithms to be used for character handling and filtering in content management systems.

The W3C has a good explanation of using character encodings in (X)HTML and CSS [11]. The most important rule is that the encoding should be specified in both the HTTP headers and the (X)HTML or CSS document for best compatibility.

Web browsers widely support web pages written in Unicode encodings, so it is possible to build on this feature.

2.2.3 Language Information and Text Direction

Section 8 of the HTML 4.01 recommendation [12] specifies three key attributes of HTML that allow for language identification and directionality setting. The `lang` attribute, when applied to an element, specifies a language code that defines the base language of the element's attributes and content. This is useful for a number of reasons, as explained by the recommendation referenced above:

- Assisting search engines
- Assisting speech synthesizers
- Helping a user agent select glyph variants for high quality typography
- Helping a user agent choose a set of quotation marks
- Helping a user agent make decisions about hyphenation, ligatures and spacing
- Assisting spell checkers and grammar checkers

The HTTP `Content-language` header can also be used to specify language, but HTML attributes should override the language where appropriate in a mixed language context.

The `hreflang` attribute has a similar role. It informs the user agent of the language of a resource being linked to in an HTML link tag.

Language codes in HTML were originally constructed according to RFC 1766, which was most recently replaced by RFC 4646 and RFC 4647 and are jointly referred to as BCP 47 [13]. The structure of a language code is as follows:

*language ["-" script] ["-" region] *("-" variant) *("-" extension) ["-" privateuse]*

The only mandatory part is a language code, which can be followed by an optional script name (Latin, Cyrillic and so on), a regional variant identifier (for example US and UK in English), any number of variant and extension identifiers and an optional private suffix (maintained for backwards compatibility). More information about these tags can be found in the W3C I18N article database [14].

Because different scripts are used for specific languages, it is possible that text be written left-to-right (LTR) or right-to-left (RTL) independently of the language being used. Latin scripts from several languages appeared through the years, replacing or adding to RTL written ones. Although Unicode defines a few control characters to specify direction, it is generally suggested that HTML documents should not use them and build on the related HTML features instead.

HTML provides the `dir` attribute with `RTL` and `LTR` as possible values, which allows for text direction specification. A bidirectional algorithm is specified to handle cases when RTL and LTR text is mixed, and a `<bdo>` tag is defined to explicitly specify direction when the algorithm gets to an undesired result without further instructions.

As direction is actually presentational information, CSS 2 [15] also has support for the `direction` property to specify RTL or LTR as well as the `unicode-bidi` property to affect the bidirectional algorithm.

XHTML carries both the language and direction attributes over from HTML, except that the `lang` attribute is replaced with the XML standard `xml:lang` attribute.

The W3C I18N FAQ has an extensive document [16] on text directionality.

2.3 Separation of Content and Presentation

Once we have the technological base to build on, we need to find ways to utilize it to benefit our users. When designing a web site with multilingual requirements, separation of content and presentation becomes vital to the success of the project. The key rules are the following:

1. **Use CSS extensively.** Richard Ishida [3] uses the example of emphasized Japanese text. When resorting to HTML `<b>` or `<i>` tags for emphasis, the Japanese letters need to be written in bold or italics. However the Japanese would use dots above the text for emphasis or a different background color, keeping the text itself intact. If we build on CSS, different style sheets for different languages can provide adequate display rules for emphasized text. This also helps prepare for bidirectional text presentation.

2. **Avoid text on images when possible.** Every image with text on it is an immediate target for replacement on translated versions of the page. Regardless of whether it is a part of the site design or user specified content, it needs to be translated. The web site should be designed so that images are replaceable when different languages are used.

3. **Prepare for text expansion and contraction.** It is common to design web pages or even smaller areas (like sidebars or blocks) on web pages for a specified screen width. If the width of these parts is not adequately chosen, translated versions of the text written into them can easily not fit or leave an undesired empty area. The OmniLingua Resource Center has good source data [17] on language expansion and contraction. This data shows that English to Finnish translation results in contraction up to 20-30%, while the word length increases by 10-15% at the same time. English to Spanish translation however can lead to 25% longer text. This means that if a layout design does not allow for text to expand or breaks when text gets significantly shorter, it is not suitable for multilingual needs.

4. **Think about possible cultural differences ahead of time.** When serving an international community, application of colors, alignment and imagery can have very different effects. Jakob Nielsen's Designing Web Usability has a perfect example [18] with an ad showing a switch. It is turned downwards and the ad says: "Turn this on for more information." Nielsen notes, that if a switch is turned downwards, it means it is already turned on in many countries around the world. Although cultural differences are not a focus of this thesis, it is important to think about these differences when planning content.

2.4 Multilanguage Interface and Content

Single language web site builders are in a convenient position to build a system in their native language and post content in the same language. However when building on an existing system, most of the time an English-based engine is working behind the scenes. This is because English is the most common language used by developers around the world, and thus has become a de facto default language for CMS products. Creating a single language web site in a different language with such a system could immediately become difficult, if internationalization is not taken into account in that product. Going further into the requirement of having a multilingual interface and content opens up new layers of required features. The interface can consist of built-in text provided by the system, as well as input provided by the administrators (site name, menu items, disclaimers, etc.).

2.4.1 Types of Foreign Language Based Web Sites

I asked the Drupal community to provide use cases of their existing and planned internationalized web sites in 2006 [19]. Going through the data provided and filtering the comments, I have identified the following practical use cases for web sites built with an English based system but in need of support for foreign languages.

English (factory default) only This is the simplest use case. In fact it means that the English "factory default" text can be used in the project and that English content is posted. User specified interface text is in English. This monolingual scenario is the simplest, and is always supported in every system.

English (customized) only When one needs a customized English language web site (different site design or different wording for text accounting to US and British English differences or stylistic requirements, for example) it is still quite close to what the system provides by default. Only some text and design elements need to be changed. User specified interface text is in English.

Single foreign language only This monolingual scenario is taken into account when a web site is built with a system, but the factory default language should be completely replaced. This requires that the text of the interface be completely translatable and that the language of the resulting site be configurable so the generated web pages show the right content with the proper language code. User specified interface text is given in the actual language used.

Multiple interface languages only On a photo showcase site or an external data based web site (like a search engine) where content is not a target for translation, it is still a common requirement to allow the interface to be presented in various languages. Users should have the possibility to choose the desired interface language, and the system might choose a reasonable default for the user when visiting the site for the first time. User specified interface text is given in all languages used.

Multilanguage content on the same site, not associated Multilanguage blogs and international community news sites are typical examples of the use case when posting of content in multiple languages is a requirement, and these posts remain stand alone pieces and not connected to each other (as translations of the same content). Content needs to be marked as being in a specific language selected from multiple languages. Interface language availability for the same languages might be a requirement when building such sites, in which case user specified interface text is given in all languages desired.

Multilanguage content implemented as sub-site Many big international companies have regional sub-sites for their local businesses. However these sites use a slightly different page layout and design elements, and often have a distinctively different structure of pages. For example, these sites do not require the ability to jump to the driving directions page of the German office from the driving directions page of the French office, especially that there is no requirement that both pages exist and are in a similar content structure. This kind of site design allows for the best local adaptation, but does not allow for content to be related between the sub-sites. User specified interface text is managed uniquely on all sub-sites.

Multilanguage content with translation association The most complete approach to multilingual site building is to have copies of the same content in different languages that are linked together so the system can show the user an initial version and then the user can choose a different translation if required. In this use case, the system can show the appropriate interface for the content language desired. The main challenge is that if content is not available in the desired language, there is no direct answer to what should happen: a fallback to some other language, an error message and a redirection to a search page are all possibilities. A system should support different approaches. The design of such a system allows for complex workflows for site administrators and content authors too. Translations of the same content can be monitored for timeliness and multiple language versions of the same content can be required to be written before a text is published, for example. User specified interface text is given in all languages.

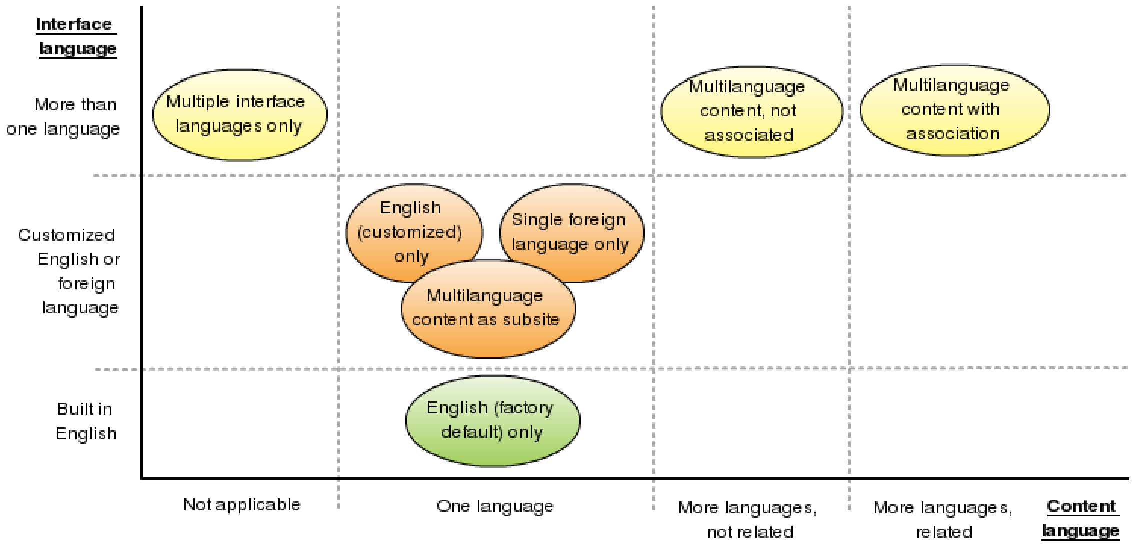

Interface language
More than one language
Customized English or foreign language
Built in English
Multiple interface languages only
Multilanguage content, not associated
Multilanguage content with association
English (customized) only
Single foreign language only
Multilanguage content as subsite
English (factory default) only
Not applicable
One language
More languages, not related
More languages, related
Content language

Figure 2.1: Types of foreign language based web sites

As the figure shows, sub-sites actually have no multilanguage requirements and as a result, workflows and permissions are not related to available languages. This is often a practical way to side-step issues with multilingual content handling but results in a weak user experience and uncontrolled editorial flow. It should be noted, however, that any of the site types involving one or more content languages or at least one non-default interface language require internationalization. Although it is possible to build sites with requirements in most other parts of the type matrix shown above, only the types shown are relevant in this thesis.

Naturally actual web projects often move between these models, so an ideal system should support seamless adaptation to the chosen type of site.

2.4.2 Distinguishing Interface from Content

In traditional desktop applications it is quite straightforward to translate the application interface. In open source systems, this traditionally involves the following steps:

1. The translator runs an extractor program on the source code, which typically generates a text based file in a standard format with interface strings found in the code. Alternatively, programmers can place identifiers in the source code, and a resource file can define the strings for the identifiers.

2. This file is loaded into a special program or a simple text editor, and text is translated to the required language. A complete translation is saved.
3. Packages of the translation are distributed and – after imported or installed – can be used with the application.

With web site management systems, however, things are very different. First, there is a set of application provided interface elements in some systems written in the "factory default" language, as mentioned above. Then there are possible modules or plugins added to the system with their own interface elements. Site designers refine and extend the built-in interface of the application to be better suited for the actual web site's needs. Finally there are interface elements specified by the site administrator. A prime example of these are menu items and other navigational helpers, which are required to be customized by the maintainers of a web site.

The application and plugin provided user interface elements are possible to be treated like classic desktop applications, translated via an external file generated with an extractor program. The advantage of this approach is that translation teams can provide language files before someone builds a multilanguage site with the system.

Custom web site design elements and site administrator specified text and images are by their nature specific to the actual project being worked on. In the first case, where the custom elements will not change often, we can reuse the desktop application workflow, but with site maintainer specified text and images, a web based frontend should be provided for the comfort of the user.

Distinction of interface from content is especially important in some of the use cases described above in which the web site might use different languages for content and interface presentation. Higher level tools for interface and content translation are discussed in the next section.

2.4.3 Translation Friendly Composite Text

When working with interface text, translating complete literal sentences is fairly straightforward when given a simple mechanism to look up a translation for a specific language. However, composing text of variable parts brings a few issues worth examining. While working on better translation support for Drupal and by looking through recommendations, I have identified the following common issues (examples in PHP, the imaginary `translate()` function plays the role of a lookup function for translations, working in the context of the current language used):

1. **Composed text should not be translated as a whole.** Once we put variable text segments into the composition, we end up with a potentially endless number of strings to translate. In case we use `translate('You have chosen '. $type)`, we will have an infinite number of strings for translation, because `$type` could be anything and is only filled in runtime. Sometimes a set of possible `$type` values can be collected, but if we think about concatenating numbers into strings the same way too, that leads to an even worse situation.

2. **Different word ordering should be supported.** If a system does composition with exact variable placement, it is not going to be easily translatable. If we write: `translate('You have chosen '). translate($type)` using string concatenation, the translator has no way to reorder the words in the sentence, although grammatical rules would enforce reordering in several languages. This is still somewhat better than the solution in the previous example, but using `translate('You have chosen %type', array('%type' => translate($type)))` would be ideal, given that `translate()` supports replacement of `%type` to the specified value.

3. **Plural forms of languages are different.** When displaying counts of things, English has the simple rule that "item" is used when there is only one, and "items" is used when there are multiple. Other languages, however, have more complicated rules for plurals. Polish and Russian languages have three types of plurals with different rules for when each of them is used. A sample from the Polish Drupal Aggregator module translations [20] shows an example of how are these used.

Expression	Translation
if $n == 1$	%n element
else if $n\%10 >= 2$ and $n\%10 <= 4$ and $(n\%100 < 10$ or $n\%100 >= 20)$	%n elementy
else	%n elementów

Table 2.1: Polish plural forms example

The software should be aware that different plural form rules apply to different languages and should support the available format. To be able to use this knowledge one needs to use a special translation function, which I will call `translate_plural()` here. The usage of this function could be: `translate_plural($count, '1 item removed', '%count items removed')` to provide a sensible default for English, yet make it possible to use plural forms.

4. **Specific contexts might need different translations.** For example “off” in `'Turn off the %object'` needs different translations in Hungarian depending on what the object is. If the sentence says “Turn off the light,” then the correct Hungarian translation is “Kapcsolja *le* a világítást”, while in the case of “Turn off the TV,” “Kapcsolja *ki* a televíziót” is the only appropriate translation. Similar problems arise with languages having different articles. `'The %fruit discount expires tomorrow.'` would have “the” translated differently to Hungarian, depending on what the value of `%fruit` starts with. The “alma” (apple) fruit would need the “az” article, but the “körte” (pear) fruit would only allow for “a”.

2.4.4 Content Creation Workflow

When building a multilanguage web site, there are different workflow requirements depending on the type of the site being built. Even on a monolingual site, content editors might need to read through and edit text before it is published. When multiple languages are taken into account, this adds an additional layer of complexity.

Users should be able to specify the requirements for their desired workflow, and the system should be able to support these requirements and execute the workflow. Some content might need to have translated counterparts (like news articles on a company web site), while other content will definitely not have them (like forum posts and comments). The user should be guided to translate content easily when required and should not be bothered when translation is not an option.

A more complicated Belgian government use case in my research [19] showed that sometimes text must be available (and approved) in a set of languages before any piece of that content set can be published. In this use case, the official languages (French, Dutch and German) should have the content available already, before it can go to the live web site. A professional grade system should allow for such complicated workflows to be built.

2.5 Translation Outsourcing Solutions

Different systems store and use content and interface translations in incompatible ways, so there is a natural need to integrate these solutions. To do this, a data interchange format supported on both ends is required. Translation support should integrate with external professional translation tools, including automatic draft translation services, translation memories, and spell checkers, and should do this by supporting common formats.

2.5.1 Gettext

Most open source applications implement an interface translation system based on GNU Gettext [21], which became the de facto standard of interface translations. Three file types are supported by the Gettext tools:

Portable Object Template (POT) Text file with source message strings (usually in English). It can be used to start a new translation or update previously done translations with new interface text from the application.

Portable Object (PO) Text file with source messages translated to a specific language. Some applications can directly import and export Portable Object files, while others need a binary representation.

Machine Object (MO) A binary (compiled) representation of a Portable Object file. As with compiled programs, editing of Machine Object files directly is not possible.

Several tools exist to generate POT files and facilitate the translation of these into given languages. When new software releases are published, new templates are generated and the previous translations are merged with these templates, forming the base for updated interface translation. Gettext only supports pairs of strings or at most language specific plural formula usage, so it cannot be efficiently used for content translation interchange, which would involve large amounts of strings and other related media.

2.5.2 Computer Aided Translation Tools

Computer Aided Translation (CAT) is the process of supporting translators in reusing previously translated text for new works, as well as archiving their current work for the future. The Localization Industry Standards Association (LISA) [22] maintains a working group, which developed the Translation Memory eXchange (TMX) format as a vendor-neutral open XML standard. The OASIS XML Localization Interchange File Format (XLIFF) builds on TMX, defining a markup format and interchange language for localizable data, allowing interoperability between tools. As of this writing, TMX 1.4b and XLIFF 1.2 are the actual stable recommendations.

The philosophy behind CAT based workflows is to extract resources from native formats into a common standard localization format that is easier to build tools for. While Gettext is ideal for interface translation based on application source code, Java property files and HTML content are among other popular formats that need tools for translation

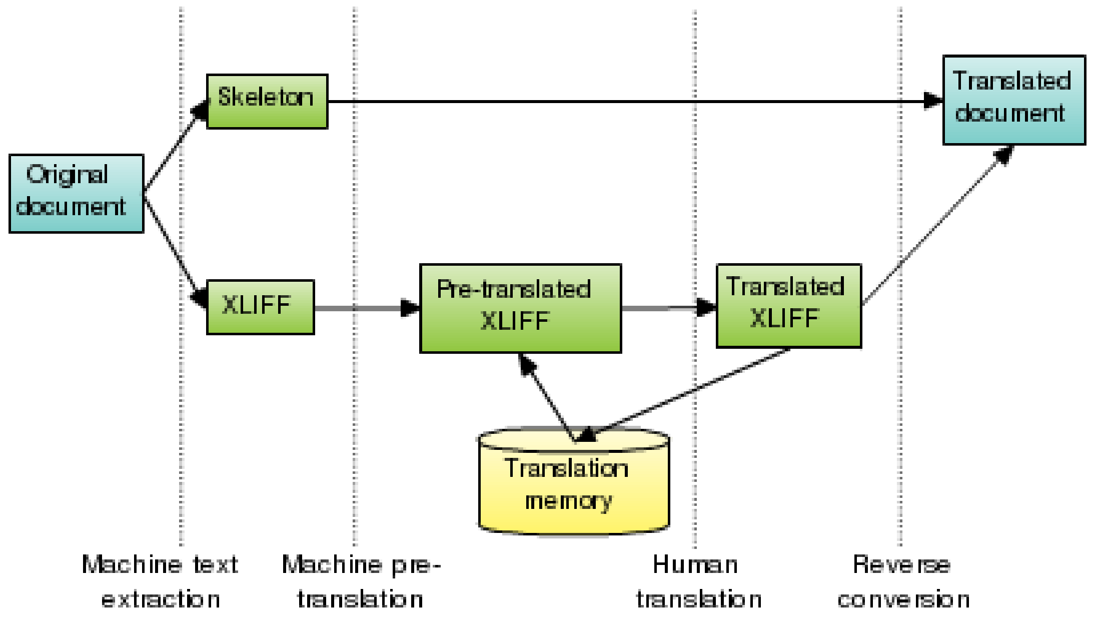


Figure 2.2: Computer Aided Translation workflow with “minimalist” approach

and a common translation memory database to reuse. Using XLIFF, the translated resources are merged back into their native format when the translation is complete, and the results are stored in a translation memory. Filters and specifications for converting to and from XLIFF have been developed for a number of file types, including Gettext Portable Objects, HTML and Java property files. Of course, not all these formats support the complete spectrum of XLIFF features, but the goal to not loose important translation data along the way is met by these mappings.

There are two types of mapping methods to choose from: a “minimalist” and a “maximalist” approach, as referred to by the XLIFF standards. These differ in how markup information is retained throughout the translation process. The minimalist approach requires a skeleton generated from the original document and only the translatable resources extracted to XLIFF (possibly with some inline markup). Inline markup cannot be removed completely because translators need to know where links and formatting appear in source documents, and they need to be able to insert equivalent markup in the translations they create when the target language requires it. With the maximalist approach, however, all structural and inline markup is encoded in the XLIFF document, and no skeleton is used.

XLIFF has a small number of generic tags used for mapping markup from any type of source document. The rules of the machine text extractor define what approach is used. In case of the minimalist method, placeholders are used in the skeleton to identify relations with parts of the XLIFF file. The extracted text is pre-translated from the previously collected translation memory, then reviewed and fixed by a human translator. The resulting translations are stored in the translation memory and a reverse conversion takes place to generate the translated document (possibly using a skeleton if available).

There are cases when the machine extractor would not be able to automatically identify translatable text or would offer parts (the author name of a document for example) erroneously for translation. A similar problem is that most source formats do not allow placing notes into the documents to instruct or help translators. Therefore the World Wide Web Consortium (W3C) developed a recommendation to aid machine extraction. The Internationalization Tag Set (ITS) [23] recommendation is a fresh development (reached the recommendation stage on April 3, 2007) that specifies a common set of tags for XML based formats to mark parts of the documents as “not for translation”, or “written in a right to left script”. ITS also allows for placing notes for translators and marking up terminology for glossaries.

While XLIFF/TMX (and hopefully soon ITS) based solutions are reusable and basically became an industry standard with the most professional tools like Systran [24] and SDL Trados [25] using them, only a few open source solutions exist to leverage a CAT based workflow, and these are not widely deployed.

2.6 The Scope of My Thesis

Although there are several key issues around multilingual web sites, a more focused scope should be defined for this thesis. As the assignment instructed, I will look into content management systems. Higher level language management, multilanguage content, interface translation support and translator workflow features are in my focus. These form a set of technologies that enable the internationalization of products. Both the user interface for these features and the implementation are important in designing a solution that fits the types of multilanguage sites outlined in this chapter.

Chapter 3

Popular Systems Used for Multilingual Web Sites

To find implementation ideas and a target platform to work with, I looked through some of the most popular open source content management systems used to build multilingual web sites and examined their approaches to storage, workflow and display of multilingual text.

3.1 Joomla

Joomla [26] is one of the most well known open source CMS solutions. It is regarded as one of the most user friendly tools and has won several awards including the Packt Publishing Open Source CMS Award in 2006 [27]. Given that the success of a good multilingual system starts at the user interface, Joomla was a logical choice to look into as a possible solution. As of this writing, Joomla 1.0.12 was the latest stable version (1.5 being in the beta stage) and the one I have worked with.

3.1.1 Included Language Support

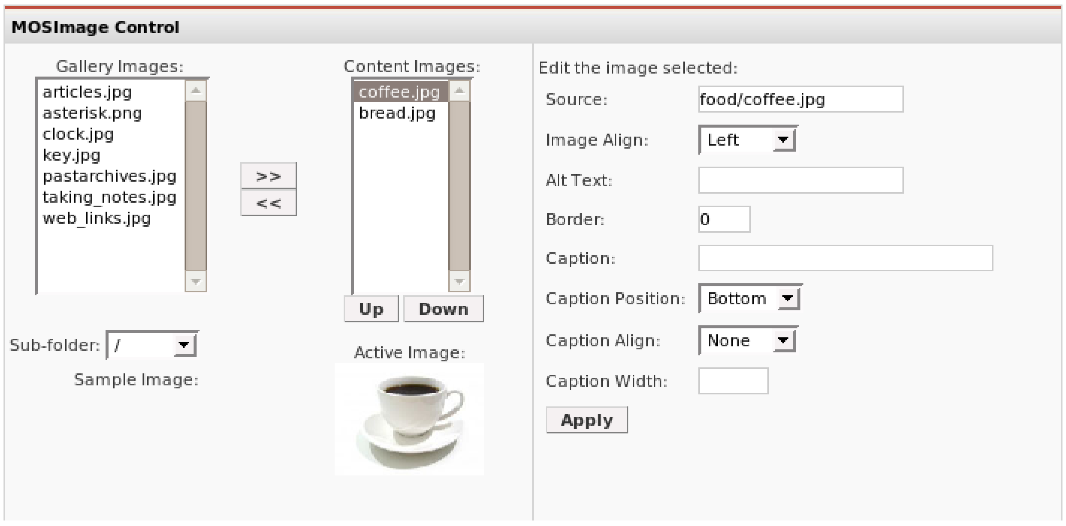
Joomla has interface language translation support included in its default installation. This allows for uploading pre-created packages of translations, which get saved into the file system and offered to the administrator to help set the interface language. This system only allows the upload of pre-created language packs, and adding a new language is not possible without installing a translation at the same time.

Interface translations are defined through PHP constants and composite strings are specified with placeholders in the `sprintf()` format, like `"Please enter a valid %s"`. Different plurals are not supported, but the order of placeholders can be changed thanks to `sprintf()`.

3.1.2 JoomFish

JoomFish [28] (created by Alex Kempkens) is the official multilingual content support component. It adds a general translation layer on top of the Joomla database handler. I have examined JoomFish 1.7, which is compatible with Joomla 1.0.12. The configuration of this component depends heavily on the actual database tables and fields, so web based configuration is not possible. XML based configuration files set the translatable table fields and allow for the translation of specific parts of the database.

A generic web based editor is provided to type in translations for these fields. Because only text based fields can be translated, simple text editors are provided. In case there is structured information stored in a text database field (like an image file name with metadata), the user must know the structure and be sure not to break its value when editing. Helpers are not included to empower the user. Only single table data is editable, and relations of data (like content to category relations) are not possible to modify.



MOSImage Control

Gallery Images:
- articles.jpg
- asterisk.png
- clock.jpg
- key.jpg
- pastarchives.jpg
- taking_notes.jpg
- web_links.jpg

Content Images:
- coffee.jpg
- bread.jpg

Sub-folder: /

Sample Image:

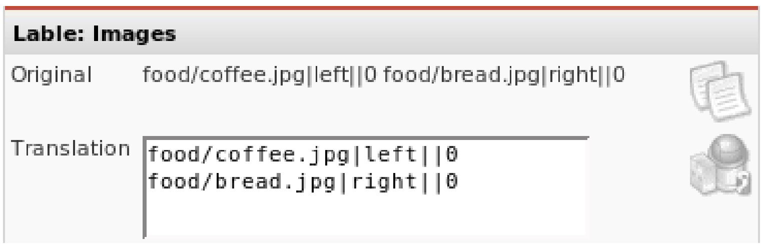
Active Image:

Edit the image selected:

Source:	food/coffee.jpg
Image Align:	Left
Alt Text:	
Border:	0
Caption:	
Caption Position:	Bottom
Caption Align:	None
Caption Width:	

Figure 3.1: Image control on the original content editor page (arranged horizontally)

As the figures show, the image selection control on the content editing page allows the author to browse previously uploaded images, select a list of images used in the current post and preview the selected item. Images can have alignment, alternate text and caption properties set. Joomla provides a rich editing interface for these details.



Lable: Images

Original food/coffee.jpg|left||0 food/bread.jpg|right||0

Translation
```
food/coffee.jpg|left||0
food/bread.jpg|right||0
```

Figure 3.2: Image control on the translation page

However, translators have some difficulties getting an editing area with the serialized information of images. Because the JoomFish translation system is as generic as possible, it does not know that an image editing user interface should be displayed here and it only knows that a specific field of a database table should be edited.

On loading Joomla, the implementation of the JoomFish component replaces the global database layer with its own database abstraction layer. The multilanguage database layer replaces load, update and insert operations, so the translated text can be written into the dedicated JoomFish table with references to the original table and primary key.

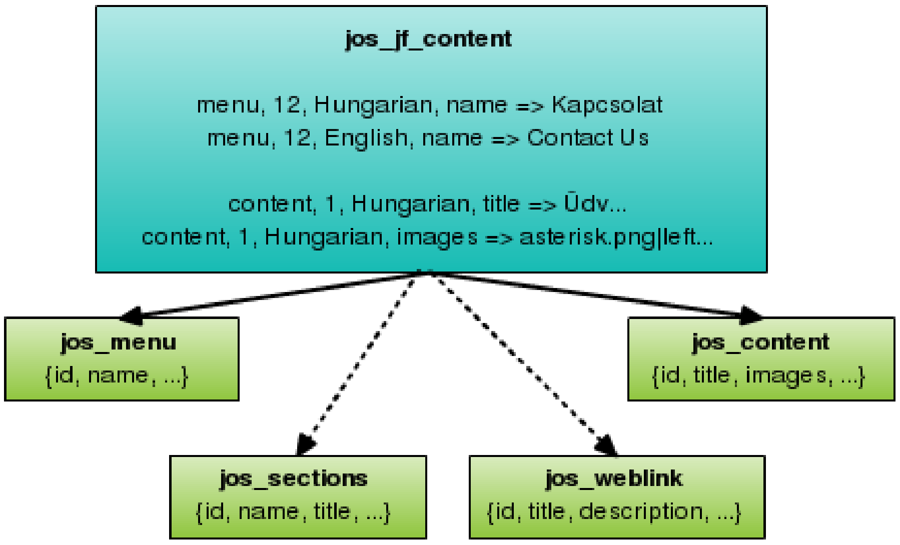

jos_jf_content
menu, 12, Hungarian, name => Kapcsolat
menu, 12, English, name => Contact Us
content, 1, Hungarian, title => Üdv...
content, 1, Hungarian, images => asterisk.png|left...
jos_menu
{id, name, ...}
jos_content
{id, title, images, ...}
jos_sections
{id, name, title, ...}
jos_weblink
{id, title, description, ...}

Figure 3.3: Database model of JoomFish with some sample data

When loading an object (content, menu item, category and so on), the replaced database layer loads it from the original table and the associated translations from its own table, and then rewrites the object's values to reflect the translated state. There are some fields excluded from translation, like the author names, published flags, and created dates, and therefore it is not possible to have a different author or publication state for a translation.

The language used to display a page depends on several factors, if the administrator enables automatic language switching. An HTTP GET parameter specifies the language when available, or a previously set cookie can be used if given. The HTTP Accept-language value set in the user's browser is used for language detection if a specific language is not requested. Finally the site has a default locale set if nothing else specifies a correct language.

3.1.3 Evaluation

Joomla by default allows interface translation. With JoomFish it also provides basic content translation, using a very extendable architecture that can be configured for any database table with simple XML files. This general approach has some drawbacks, though: it is not possible to conveniently translate non-textual content, only a given set of properties are translatable on any content object, and finally the double loading of data results in a performance impact on the database. Unfortunately, Joomla does not come with any tools to support a CAT workflow and XLIFF support is scheduled for JoomFish 2.0 at the earliest [29] (currently 1.8 being under development).

From a visitor's point of view, the most problematic aspect of Joomla is that the language code is not kept in the URL. Once someone visits a page with a language code in the URL, a cookie is set with the language code and then shorter web addresses are used. This makes it impossible to send links pointing to a particular version of the content and index different language versions by search engines.

Another basic problem with the Joomla system is that it tries to use UTF-8 all around the web site, but the default templates specify ISO 8859-1 encoding for English and ISO 8859-2 for the Hungarian translations (although the translation files use UTF-8 characters). Evidently, encoding handling is not yet clear in the system.

3.2 TYPO3

TYPO3 [30] is one of the most complex and, at the same time, most powerful open source systems on the market. The complexity is easily shown by looking at the TypoScript declarative language especially developed for TYPO3, although the system is generally built on a PHP and database driven backend. It has built-in support for interface localization as well as the so called “multilanguage content” and “multilanguage content integration” methods for content translation. Being built-in features, there is no need to install additional components, the existing solutions are tightly integrated into the system, language controls are placed where the administrator expects them. I have worked with TYPO3 4.1, which was released March 6, 2007.

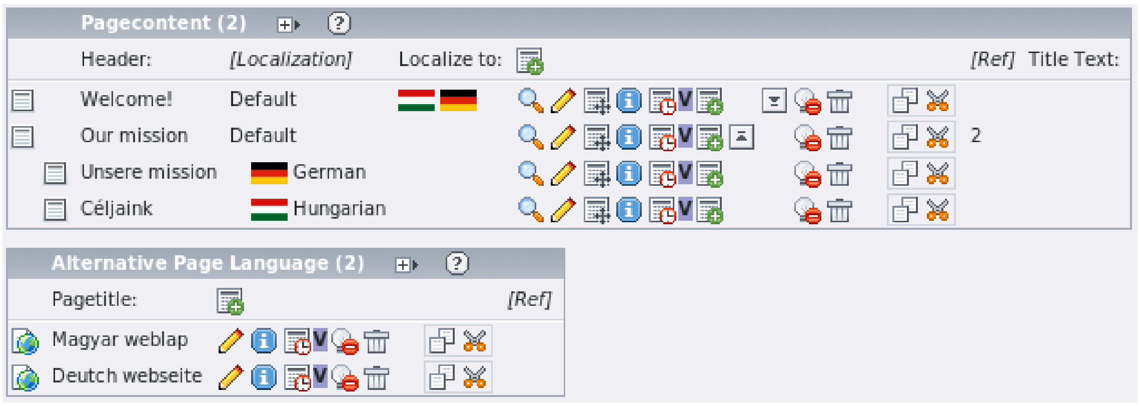
3.2.1 Interface Translation

TYPO3 implements interface translation based on a custom “locallang-XML” (llXML) format [31]. This allows for meta information and default (English) language text specification in TYPO3 modules, as well as possibly included translations in published packages. An extension named “llxmltranslate” is provided to assist translators with providing a web interface to translate interface files.

3.2.2 Multilanguage Content Method

Two different multilanguage approaches are supported. Multilanguage content and multilanguage content integration differ in how the site structure is built. TYPO3 models web sites as tree structures of web pages. It allows for translation of web sites by creating different trees (essentially sub-sites) for the translated versions. This is easily done, and also provides the extra feature of possibly having pages in one language that are not suitable in others. The problem for the end user is that it is not possible to switch languages on the site pages and therefore the language selected on the site entry page is used.

3.2.3 Multilanguage Content Integration Method


Pagecontent (2)

Header:	[Localization]	Localize to:	[Ref]	Title Text:
Welcome!	Default			
Our mission	Default		2	
Unsere mission	German			
Céljaink	Hungarian			

Alternative Page Language (2)

Pagetitle:	[Ref]
Magyar weblap	
Deutch webseite	

Figure 3.4: Alternative page languages and layers of languages for a page component

The multilanguage content integration method allows administrators to have one page tree with translations of page fields saved under the page as alternate values. This is similar to the Joomla approach with an eagle eye view, but it is implemented differently.

Every web site tree can have *web site languages* specified. These have a name and a flag associated with them. Once these languages are set up, the content created in the default language can be localized by adding *alternative page languages* below the previously created pages. This means that there should be a default language in which pages are added, and the translated contents are *layered over* the default values when displayed. The default content is used when there is no translation to a specific language.

Among other overview possibilities, a convenient tree view is provided that shows the translated content elements below each element created in the default language. The tree view mirrors the internal implementation of TYPO3, showing that the language layers reference content in the default language, and the alternative page languages define what localization can be added for a page.

Language selectors (flags) can be shown in this mode on the web site so visitors can see content available in different languages. Flags of unavailable content are dimmed. A HTTP GET parameter is used to keep track of what language is selected, and TYPO3 overlays content (menus, pages) available in that language onto the default values on all page views.

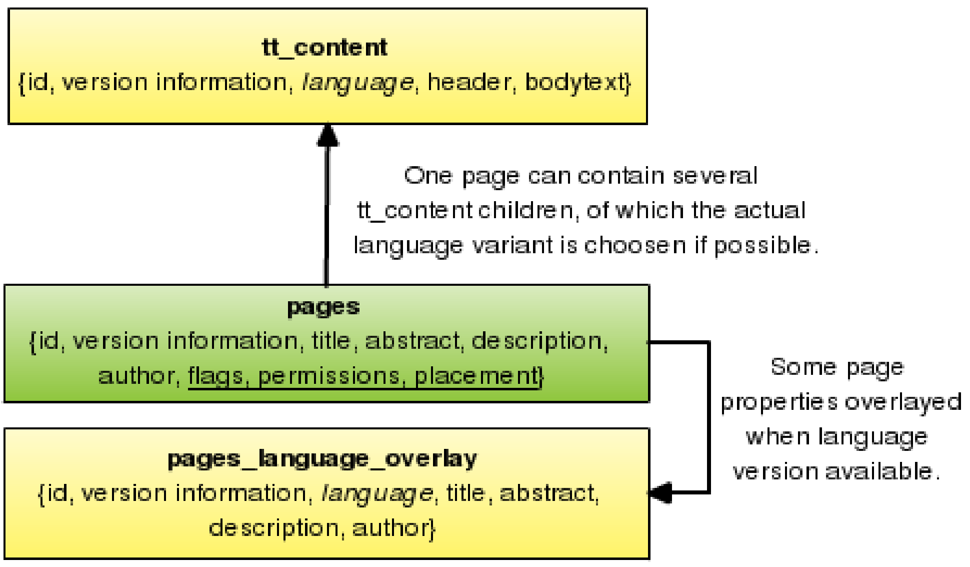

tt_content
{id, version information, language, header, bodytext}
One page can contain several tt_content children, of which the actual language variant is choosen if possible.
pages
{id, version information, title, abstract, description, author, flags, permissions, placement}
Some page properties overlayed when language version available.
pages_language_overlay
{id, version information, language, title, abstract, description, author}

Figure 3.5: Database model of the multilanguage content integration method

On the database level, the *sys_language* table contains the list of languages. The *pages* table stores the page details, while the *pages_language_overlay* table stores the overlay values for alternate page languages. This includes meta information like author name and email, creation date and content versioning information. Because pages themselves only store higher level information of displayed web pages, the content objects can be found in the *tt_content* table, for every language. When pages are loaded, the content of these tables are taken into account. Pages, overlays and content objects all support versioning.

Finally, a “Localization Manager” extension was implemented in December 2006 to better support translations’ overview possibilities as well as the import and export of content for translation. This extension uses the Microsoft Excel XML spreadsheet format for external translation support. Although this cannot be used directly in a computer aided translation workflow, Orange Translations, LLC announced [32] that it will sponsor further development of this component to integrate into CAT systems. I was unable to find any publicly available results of these efforts.

3.2.4 Evaluation

The TYPO3 system was not built with end users in mind. To set up a site even with only some simple customizations, administrators need to learn TypoScript and various objects and properties to use. This means that it is mostly popular among solution providers. While the multilanguage concepts and possibilities offered by TYPO3 are adequate for most needs, the complexity of the user administration interface and its steep learning curve does not make it a natural choice in most projects.

3.3 Plone

Plone [33] is a content management system built on the Zope platform and written in Python. One of its immediate marketing points is that it is “built for multilingual content management from the ground up”, and even has support for right to left written languages. As a prime example, Plone was chosen to power the GNOME homepage [34] partly for its strong internationalization support, so this was a good candidate to look at.

3.3.1 Interface Language Support

Plone has content language and interface language support. The administrator can set the language of any content on the site and new content is created as language-neutral. There is a predefined list of languages available in Plone from which the administrator can choose.

The PlacelessTranslationService extension allows for translation of the interface of Plone sites. Precreated translation packages are available in the PloneTranslations extension (as Gettext Portable Object files).

Finally, the PloneLanguageTool extension can handle automatic language switching. It allows for the setting of a list of languages actually used on the site (a subset of the list of predefined languages available in Plone). A flag (or language name) list is generated for visitors so they can choose from the translations of the current page in those languages, if available.

Plone also supports different language negotiation schemes. The language can be specified in the URL, in a cookie, or in the Accept-language browser setting when these are available, and can be the site's default setting otherwise.

These tools are still not enough for content translation, and only negotiation and user interface elements are supported by these extensions.

3.3.2 LinguaPlone, XLIFFMarshall

LinguaPlone is what the Plone developers call the third generation of multilanguage support in Plone. The previous generations' solutions had different approaches, with the second generation being similar to what Joomla and TYPO3 implements. The third generation LinguaPlone tool, however, acknowledges the limitations of the previous approaches in document workflow, FTP and WebDAV import and export support and compatibility with other existing Plone components.

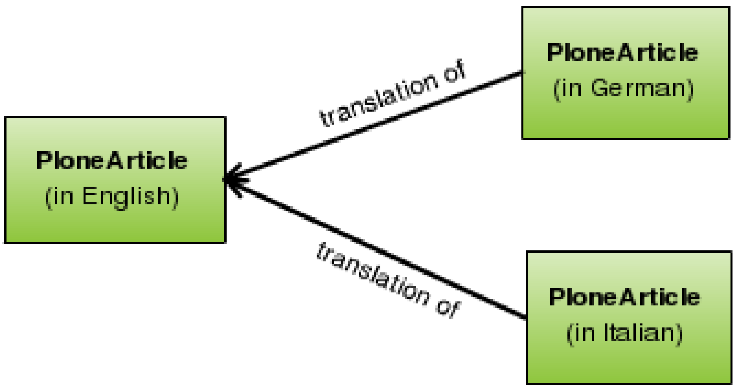


Figure 3.6: LinguaPlone handles translations as first class objects having relations

For these reasons LinguaPlone implements a translation method where different content types can be language-enabled, but translation instances are stored as different first class content objects (unlike TYPO3 and Joomla). This way every existing functionality can work with translated content, and these objects have their own URLs. Language independent fields can be shared between different translation instances. A content object describing an event could have a date field shared, for example, in all translations.

LinguaPlone defines a canonical version for every content object and points translations back to that version. Language independent fields are loaded from this object but are stored in every translation with the same value so every other functionality can work with first class content objects. Property accessors of these shared fields guarantee that changes are made to all instances at the same time. This also means that when LinguaPlone is disabled, every content can still be worked with.

A two pane interface is provided to create content translations and to show the canonical version in a second column while the user is typing in translations.

Sasha Vinčić gone even further and implemented XLIFF import and export support in the XLIFFMarshall package so a computer aided translation workflow can also be supported by this package.

3.3.3 Evaluation

Plone, with its mentioned components, allows the complete translation of web sites with workflow support and provides the most comprehensive feature set for multilingual sites among the content management systems I examined. It serves as a good example for implementation in other systems, although some details, like how the accessors for shared fields are allowed by the underlying object database, would not be readily available in other systems.

3.4 Drupal

Drupal [35] is a free software package that allows an individual or a community of users to easily publish, manage and organize a wide variety of content on a web site. Sites are built with Drupal at IBM, NASA, NATO, UN, Yahoo, Sony, MTV, Canonical (ubuntulinux.com), etc. Drupal has a strong Content Management Framework (CMF) foundation that enables it to meet different content management needs.

As of this writing, the 5.1 version of Drupal is the latest stable release, so I used that version as a basis for this comparison. While interface language support is built into the system, content language support is only possible with additional modules. There are two similar module packages built for this task.

3.4.1 Interface Language Support

Drupal has built-in interface language support through the locale module. This module allows administrators to set up a list of languages to make the interface available in. The system collects untranslated text on the fly, which allows for web based translation of the interface. It is more convenient, however, to download a pre-translated package of Gettext Portable Object files and import them into the web site's database. Drupal delivers web pages in different languages based on user settings with anonymous users accessing pages in the site's default language. The Gettext PO based translation method allows for the reuse of several open source Gettext tools.

The interface language support, however, only spans to "factory built-in" strings. User specified interface elements (menu items, the site slogan, and so on) are not possible to translate.

3.4.2 Content Translation Support

Different objects (menus, categories, site blocks, web site properties, user specified content, comments, etc.) are stored and handled differently in Drupal. Every type of object has dedicated functionality and storage methods, as is the case with Joomla. This means that the translation of a web site does not stop with translating content. As a consequence, different objects might need distinct translation methods to match their purpose.

Because Drupal 5 has a good base of language settings, both well known module sets build on this capability. Users can specify used languages on the locale module interface.

3.4.3 "Internationalization" Module Package

The Internationalization (i18n) module package, developed and maintained by Jose A. Reyero, is the classic choice when building multilanguage sites with Drupal. As of this writing, the current set includes the following most important modules:

i18n.module Allows for language settings of content, categories, menu items and site properties. Handles automatic language selection for the user.

translation.module Stores relations of content and categories, so translations of the same content can be represented.

i18nblocks.module Provides meta-blocks for multilanguage block availability. This enables administrators to modify block properties all at once for different translations.

i18nprofile.module Implements translation support for user profiles, so profile details can be asked for in the user's language.

The i18n module set by default takes a similar approach to Plone by storing objects in different languages separately and forming relations between them. This way translations of the same content can be shown to the user. Unfortunately, this results in a sometimes unnecessarily cluttered interface. Many users are not interested in dealing with special meta-blocks for block translation or adding translations of categories in separate instances so they can be related as translations of each other. For this reason there are newer replacement modules in the i18n module set that allow for lower level translation of some objects (menu items, taxonomy terms and generic strings), only allowing "overlays" of textual properties. Having both approaches implemented allows users to select what fits their needs on a case by case basis.

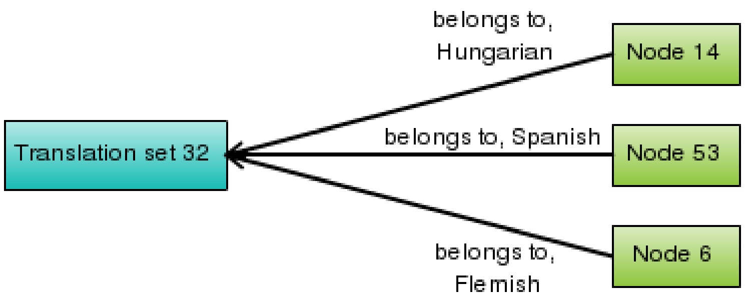


Figure 3.7: Content instances are related to a translation set in i18n module

3.4.4 "Localizer" Module Package

Localizer was born from some of the frustrations mentioned above with the sometimes overwhelmingly complex i18n module interface. It was largely built on the i18n module code base (and also brought some concepts from the translate module built by Rob Ellis), and is developed and maintained primarily by Roberto Gerola. Localizer includes the following modules:

localizer.module Provides a general language setup interface, as well as language selection. A generic string translation mechanism is provided.

localizerblock.module Adds a language field to blocks (but no general placement helpers like i18nblocks module's meta-blocks).

localizernode.module Implements language support on nodes, as well as translation relationships between them, to supply source data for language selection.

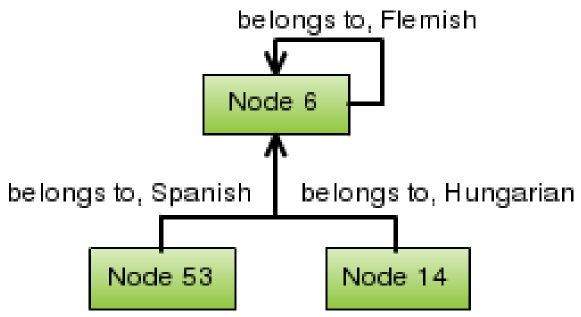


Figure 3.8: Content instances are related to a parent content in localizer

There is also a set of modules built on the generic translation mechanism provided by the localizer module. This mechanism is modeled very similarly to the Joomla approach. Object names and object keys identify a record in the database (like "menu_item" and the item identifier). An object field specifies what field is translated from that record. Menu, categories and site properties are translated using this approach. Node translation is similar to what the i18n module implements, although there are some additional limitations due to content instances not related to a translation set but rather a parent content item.

3.4.5 Evaluation

Drupal comes with two different module packages for multilanguage web sites. While both of the approaches have a Plone-like content translation method (having different instances for translations), other Drupal objects are handled with an approach closer to Joomla in the localizer module package. Extension possibilities are offered by Drupal so that these modules can plug into database query building and interface generation. Still Drupal 5 by its nature is built for single content language web sites primarily, and the solutions used in the mentioned modules sometimes need to work around awkward limitations. It is also apparent that because multilanguage handling is not a core value of the system, third party contributed functionalities need to be taken care of in the language supporting modules.

Chapter 4

A Comparison of the Examined Solutions

The following section compares the features I will look at based on my findings of multilanguage web sites' requirements, according to the definition of key areas I presented in the second chapter. Later I will present the reasons why I choose Drupal for my implementation.

4.1 Language Management and Detection

All examined systems (with the tested extensions) provide decent language management features. A set of languages used on the web site is defined and a selection algorithm is configured on the web interface so that the software can select a language to use to show content and interface for the visitor. The following factors can be configured in all systems:

1. **Language specified in the URL.** If a specific language is asked for in the URL (either in the domain name or path), it always overrides any other preferences.

2. **User language setting.** Most systems allow the user to set a preferred language in her profile, which is used on subsequent visits.

3. **Language cookie remembered.** For anonymous users, the previously viewed language is remembered in a cookie and used to return to that language on the next visit.

4. **Browser language detection.** Browsers send an HTTP Accept-language header with information about language preferences (if the user sets this up in her browser). If a language preferred by the user is found in the list of available languages on the site, that one is selected.

5. **Website default language.** Every software examined supports the notion of a site default language. If nothing else identifies the language, this provides a last chance fallback.

Configuration of language detection involves selecting a number of the above factors (and in some cases the order).

Joomla unfortunately suffers from a problem in this area. By allowing to change the language with a URL parameter but then remembering that language outside of the URL, there is no sign of the language variant displayed on the page in the web address itself. This makes it impossible for search engines to index multilanguage content and users to post links to specific language versions without a deeper knowledge of how Joomla works. Plone, on the other hand, enforces language prefixes (and page hierarchy mappings) for URLs, thus making the best practice the convenient default behavior. Drupal's localizer module is the only one to support different domain names for different languages out of the box.

4.2 Interface Translation

The examined systems provide a default web site interface that includes dialogs with users, navigation aids and more. Translation of these components is the first step in providing a multilanguage web site.

Drupal provides the most control to site administrators, although only slightly more than what Plone offers. Local language variants are easy to create and edit on the web interface without knowledge of the underlying Gettext tool set. Import and export to these formats is supported seamlessly. Joomla's constants based approach is limited by its lack of plural forms support, while TYPO3 requires custom tools and know-how to handle the llXML based localization files.

4.3 Content Translation

As discussed previously, every site manager defined text and media is considered content. In this respect, there are three distinct approaches identified to multilanguage content:

1. **Separate objects for translation.** This method works by storing translated versions of content objects as separate instances, relating them to each other. Plone uses this method, as does Drupal with its modules for node based content storage. Plone implements this based on the underlying object database, so access to common properties of translations can be managed. A canonical content object is defined for every such content group so a canonical version of shared properties can be managed. Drupal's modules have no support for shared properties so unfortunately this problem is stepped over. The Drupal i18n module also uses this approach for other content objects to allow different site structuring and setup for specific languages.

2. **Overlays on content objects.** Some objects can have a defined set of their properties "overlayed", in effect replaced by translated values upon loading. Shared properties are implemented by not allowing some properties to be overlayed and by falling back on the original values when no overlayed value is available. TYPO3 uses this approach, and some modules in the Drupal i18n module set also make use of such a solution.

3. **Generic database level value overlays.** A more generic implementation of content object overlays involves allowing property overlays on the database level. Database table names and key values specify a record, and a column name specifies a value to be overlayed for an actual language. This approach as used by Joomla and in part by Drupal's localizer module, is generic enough to allow for any kind of translation on the relational database level.

By looking at the history of Plone multilanguage support [36], we can see that developers around Plone tried almost all of the above approaches and ended up with separate content objects in their third major iteration. The most compelling reason for this is because there were so many tools for content objects implemented already. Version and change tracking, permission handling, workflow support, import and export functionality, FTP and WebDAV interface and others. By storing every relevant content property in every translation, even if LinguaPlone is turned off, the content objects are still present and usable by the system.

The overlay approaches are based on the assumption that translated content is really just text replacement. When the above mentioned feature set is required, reusing existing functionality built into the system should take precedence and separate object instances should be used. It should be noted that not every content type requires workflow support or language dependent permissions. A simple poll published in all languages on a web site and translated with an overlay method would collect the votes for all users in the same data store, as well as prevent users from submitting multiple votes in different language interfaces of the site. Here, the reuse of existing poll related functionality might take precedence. It is important to consider these arguments in the actual case when implementing a multilanguage solution.

4.4 Permissions and Workflow

While it is possible to translate content in every system I examined, permissions related to translations and complex workflow support differ.

Joomla offers a fixed list of user groups from which translators need to be at least in the "backend administrator" group to access the translation interface. Unfortunately, this gives them many other rights on the administrative screens. JoomFish maintains a hash value of the original text of content being translated, so a basic workflow is supported to identify stale translations. There is no versioning support for translations (neither for the original content itself). A CAT based workflow is not supported.

TYPO3 allows limiting users to specific languages and specific editing interfaces so translators can only work on their assigned languages with their assigned tools. Version tracking for content and translation overlays is supported so the system identifies and warns translators when specific details of base pages or page components change. The original and new version are shown to the translator. Although CAT supporting tools are not yet implemented, there is support for export and import of Microsoft Excel XML spreadsheets, which allow for external translation.

Plone comes with a mature workflow engine that controls the states of the document through a publishing workflow that includes states and transitions. States could include created (initial state), pending (waiting for review), published, and so on. Distinct permissions are also maintained for each state so different user group members can only make modifications they are entitled to make. Additionally, scripts can be run on transitions to inform translators and editors of changes. Unfortunately, there is no built-in workflow tailored for translations, but workflows can be configured on the web administration screen. There is an XLIFFMarshall extension available to plug the onsite workflow to a CAT based translation system.

Drupal supports user role based permissions out of the box, and it also comes with a per-content permission backend for which extensions provide the user interface. Although no notion of workflow is supported in the system by default, the i18n module includes a simple publishing workflow where authors and translators can attach states to their documents. Transitions with scripts (as in Plone) are not supported, and the localizer module has no similar feature. There is a workflow and an actions module available, though, to implement complex workflows with transitions and scripts (here called actions). There is no sample workflow provided for translations and neither of the modules examined provide actions for these workflow support modules. I was unable to find a CAT based workflow support tool for Drupal.

4.5 Comparison tables

Feature	Joomla	TYPO3	Plone	Drupal with i18n	Drupal with localizer
Web based language management	Y	Y	Y	Y	Y
Language in domain name	N	N	N	N	Y
Language as HTTP GET/POST parameter	Y	Y	Y	Y	Y
Language permanent in URL path	N	Y	Y	Y	Y
Language for anonymous user	Cookie	N	Cookie	PHP session	PHP session
Language setting for site users	N	N	Y	Y	Y
HTTP Accept-language based detection	Y	N	Y	Y	Y

Table 4.1: Language selection comparison

Feature	Joomla	TYPO3	Plone	Drupal
Translatable interface	Y	Y	Y	Y
Technology used	Constants	llXML	Gettext	Gettext
Translation management	Import by upload, stored in the file system	In package	In package	Import to database by upload and web based editing
Reordering of variable strings	Y	N/A	Y	Y
Different plural forms	N	N/A	Y	Y
Locale (eg. date format) support	N	With extension	Y	N

Table 4.2: Interface translation and localization comparison

Feature	Joomla	TYPO3	Plone	Drupal with i18n	Drupal with localizer
Functionality availability	Extension	Built in	Extension	Extension	Extension
Translation method	Database overlay	Object overlay	Associated objects	Associated objects	Associated objects
User interface	Generic, text only	Similar to content editing	Content editing	Content editing	Content editing
Shared properties	Not overlayed, limited by configuration	Not overlayed, limited by database	With accessors, in every object	N	N
Translation versioning support	N	Y	Y	Y	Y
Permissions for translators	Only administrator can translate	Language and interface limited	Flexible, state sensitive	Role or content based	Role or content based
Workflow support	Very limited	Limited	Mature	Simple (or addon)	N (or addon)
CAT workflow support	N	Partial	Y	N	N

Table 4.3: Content translation comparison

4.6 Choosing a System for My Implementation

The findings outlined in this chapter suggest that the language and workflow requirements examined at the beginning of my thesis can be fulfilled with either a Drupal or a Plone based implementation, with Plone having the strongest multilanguage support of all the systems I have examined. At this point in time, I would recommend Plone for people evaluating a content management system on the grounds of multilanguage features. A typical project, however, involves a lot more factors to take into account before deciding on a backend.

As my thesis assignment instructed me to implement multilanguage features with an open source system, I chose Drupal because it allowed me to plan a new architecture based on my findings and provide an implementation. It also enabled me to contribute my work to the open source community, in part directly to the Drupal codebase and in part as extensions to the core system. Participating in the community also allowed me to get critiqued and corrected from time to time, ensuring that my work is useful for actual Drupal implementors.

Chapter 5

Defining Requirements for a Drupal Based Solution

5.1 Drupal Architecture

The Drupal 5 framework operates with a web server supporting PHP and a database, with MySQL and PostgreSQL being the two most supported databases. At the core of Drupal is the API provided to the upper layers: database abstraction, visitor session handling, event logging, multilevel caching, etc. Initializers decide on the amount of code loaded from the full framework, depending on what is required to serve the page request, implementing a sophisticated "bootstrap system".

The page's interface language is selected in the bootstrap process, because the page contents are different for different languages, it is impossible to serve a cached version without knowing the language.

If the bootstrap system identifies that the request cannot be served from the cache, without loading core functionality, the needed modules are loaded and the processing is handed over to the registered page handler for the identified request path. That page handler works with the database, collecting the data required to build up the page and then handing it over to the theme layer to generate a response suitable for the request.

The complete system interface is capable of being translated, but the translation subsystem is only active if the so called "locale" module is turned on. This puts the core system at a comfortable distance from translations, so if such functionality is not required, it does not hurt runtime performance.

Drupal 5 comes with an alternate bootstrap mode used for installation when a backend database is not present, so the system needs to be able to work under tight resource limits.

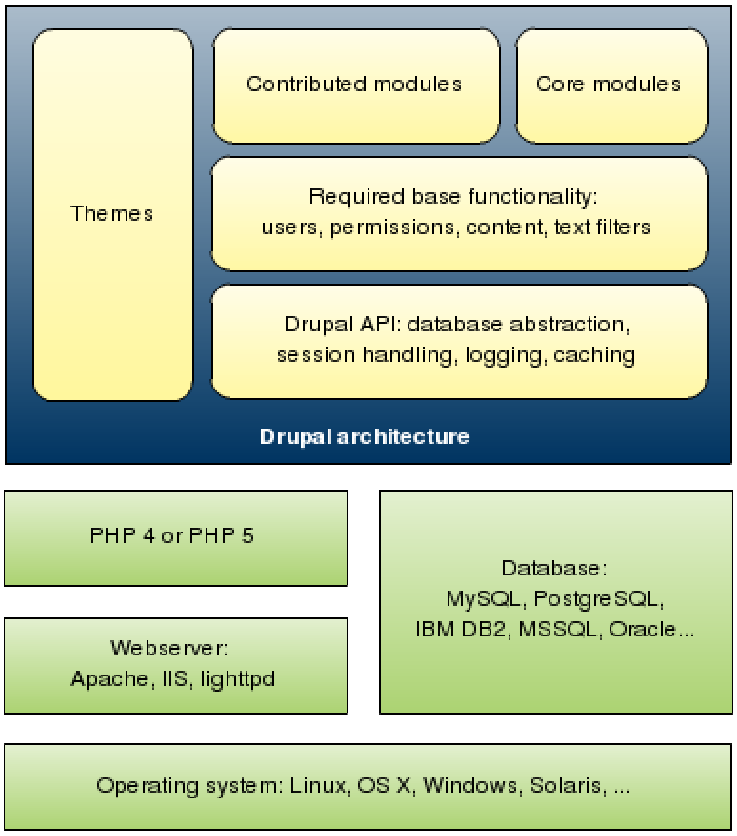

Themes
Contributed modules
Core modules
Required base functionality: users, permissions, content, text filters
Drupal API: database abstraction, session handling, logging, caching
Drupal architecture
PHP 4 or PHP 5
Webserver: Apache, IIS, lighttpd
Database: MySQL, PostgreSQL, IBM DB2, MSSQL, Oracle...
Operating system: Linux, OS X, Windows, Solaris, ...

Figure 5.1: Drupal architecture

The task of the installer is to help the user set up the database and possibly any other details related to the actual install profile used.

Drupal allows customization and extension of it's functionality with so called "hooks". Hooks allow developers to provide code to run when certain events happen, allowing modules to subscribe to the listener's list of particular events. The form altering hook mechanism allows developers to change or extend existing forms in the system. For example, it can be used to add language selector elements to content object editing forms.

Drupal serves pages with a callback registry mechanism. Modules can register their callbacks for certain path components of web addresses and when a HTTP request comes in, Drupal selects the callback to invoke based on this registry.

The view (display) layer of Drupal is separated from modules, which allows for different template languages including PHP and domain specific languages like Smarty. Modules work with theme callbacks to generate output for the client.

A deep overview of the Drupal architecture is available in the Pro Drupal Development book [37], published by Apress.

5.2 Planned Language Architecture

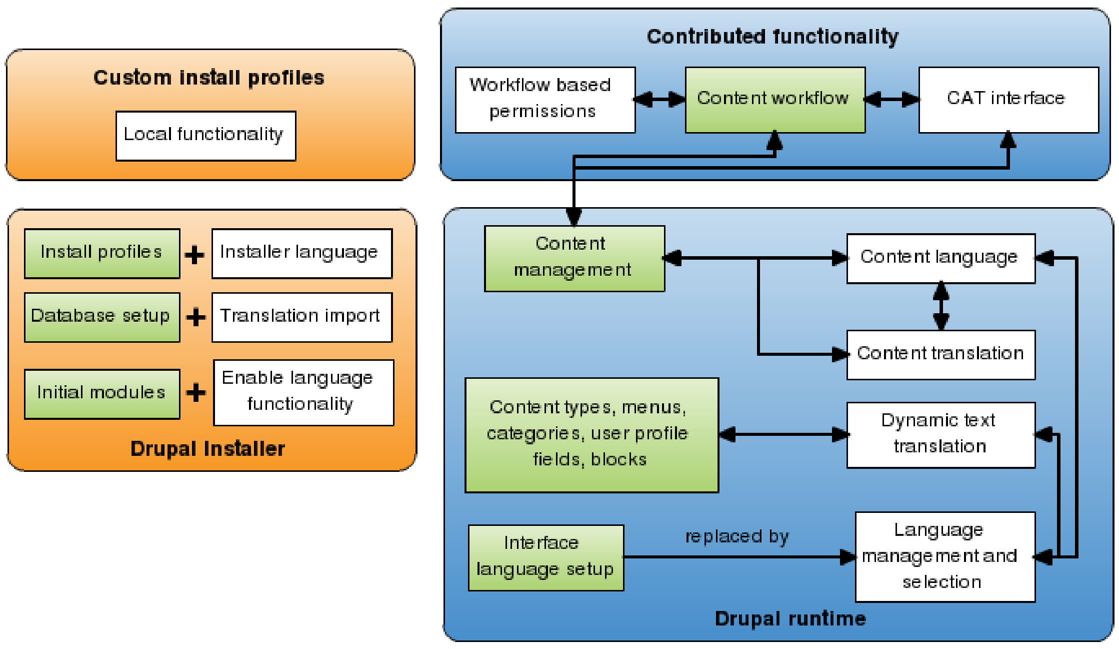


Figure 5.2: Planned Drupal language architecture

I started defining requirements and implementing some solutions when Drupal 5 was close to being released. Because Drupal only had a notion of interface language and this concept was only used in runtime, the installer and the Drupal runtime needed to be language aware to have better language support, usable interfaces for content, and user defined interface translation. The white areas on the figure show where I worked on planning and developing solutions for Drupal.

5.3 Source Code Based Interface Translation

As explained previously, Drupal has Gettext Portable Object based interface translation support. The translation templates ship with an extractor script that can be used to generate interface string templates for Drupal itself and contributed modules. External programs can be used for translation, and finally the resulting PO files can be imported to Drupal directly. The translations are stored in the database used by the system, which allows for quick access when required. Any number of languages can be set up.

5.3.1 Installer Localization Support

Drupal 5.0 was the first release to include a web based installer. Previously new Drupal site setups required the editing of configuration files and were not ready for foreign interface languages from the start. Users needed to enable interface translation functionality and import interface translations after setting up Drupal. With Drupal 5.0 a limited Drupal runtime is loaded into memory to execute the install process with a given installation profile (which can enable functionality and do database changes as required). Initially the installer was not multilanguage ready so a natural requirement was to make it capable of guiding the user through the install process in any selected language. My requirements list for this feature were the following:

1. The existing localization functionality of Drupal should be reused to allow for the same API to be used for the installer interface.

2. No database should be required, and an in-memory solution should be developed to work before the database gets set up.

3. Translators should be able to translate the installer interface with their well known tools, and translation packages should ship with the translation of the installer.

4. My solution should not hurt runtime performance of Drupal, once installed.

5.3.2 More Efficient Translation Packaging and Importing

Up to Drupal 5.0, translations of the Drupal core were shipped as one monolithic Gettext Portable Object file in the hopes that the uploading and importing of interface translations were easy for the user this way. Unfortunately, shared web hosts have resource limits on running scripts and the importing of complete translations (which can be up to half a megabyte in size) often resulted in terminated and incomplete translation imports. The second drawback of the monolithic translation files is that the text for unused functionality clutters up the database and slows down the system.

For these reasons, a more efficient packaging format and import mechanism was designed. Requirements of this component were as follows:

1. Translations of different modules should be separated, easing the work for translators as well as optimizing database usage.

2. Users should get localization functionality turned on if using a localized installer.

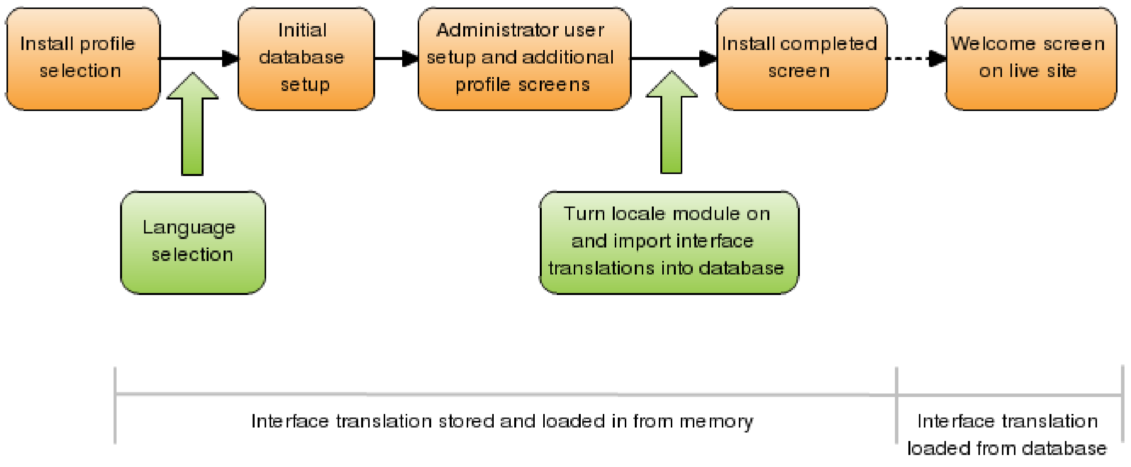

Install profile selection
Initial database setup
Administrator user setup and additional profile screens
Install completed screen
Welcome screen on live site
Language selection
Turn locale module on and import interface translations into database
Interface translation stored and loaded in from memory
Interface translation loaded from database

Figure 5.3: New functionality required in Drupal Installer

3. The localization backend should import translations for the used functionality at install time.

4. When the administrator changes the set of used modules on the site, new translations should automatically get imported and wiped as required.

5. Possible (but exactly not known) resource constraints of hosts should be taken into account.

The two tasks explained fit into the Drupal installation process by allowing language selection for the previously selected install profile, enabling interface translation functionality, and importing translations prepared by Drupal translation teams. This way the system is prepared to welcome the user in a language other than English from the start.

5.3.3 Local Functionality with Custom Install Profiles

While planning to add language functionality to the installer component, it became evident that some groups require local functionality on top of the translated interface. There is common functionality that is relevant for a local group only. An example of this was the Hungarian community, where hobbyists often set up Drupal on a shared host. Well known shared hosts have well known special setup requirements, so I decided to add supporting functionality for some of these hosts in the Hungarian installation profile. My skeleton was implemented by István Palócz and after refining the source code, I added the implementation to our installation profile [38, 39].

5.3.4 Fixing Logic Problems and Adding Smaller Features

While looking deeply into how Drupal does interface translation, a few small issues arose that required solutions. While these do not directly relate to my thesis, solutions for these issues were required to provide a complete multilanguage solution for Drupal users.

1. Language independent logging was a problem in the system. Log messages were always saved in the language the current page is displayed in, so the logging system was in need of some refactoring to log messages in English and display them in the current language used to administer the web site.

2. The menu building system was refactored as part of the Drupal 6 release and required a different menu item translation solution similar to what was planned for the logging subsystem.

5.4 Language Management Functionality

While examining the existing extensions for Drupal and analyzing the requirements posted on my request [19], it became apparent that low level, built-in language management support in is needed Drupal. The limited locale functionality only allowed for a set of languages to be specified. Directionality or native names of these languages were not known and language selection and detection implementation was not included in Drupal.

The goal of the first layer of language support improvements for Drupal was to elevate language awareness among Drupal developers by including better built-in language support. Requirements of the language management improvements are as follows:

1. Decouple language management from the locale functionality, given that language related configuration is not limited to the interface language anymore.

2. Maintain the writing direction, native names and weights of languages to be used when displaying a language or a list of languages.

3. Add web address (domain and path) based language selection, with freely configurable domain names and path prefixes. Keep in mind that IRIs should be usable here.

4. Implement a browser setting (HTTP Accept-language) based language detection to select a default language for the user when first visiting a page without explicitly asking for a language.

5. Provide a configuration mechanism for the language selection algorithm used and how it takes domains and paths into account.
6. Implement an upgrade path for earlier Drupal versions so previously set language properties are kept in the new version.

5.5 User Specified Content Translation

5.5.1 Running Multiple Sites on the Same Code Base

Drupal includes a so called “multisite” feature that allows the system to run multiple web sites on the same code base, possibly sharing parts of the database, for example user accounts. This allows for the easy set up of different web sites for different languages and could be used to provide a language selector entrance page for visitors similar to how TYPO3 implements its “multilanguage content” method. However this type of multilanguage setup is very limited, so as I declared earlier, I will specify solutions for a “multilanguage content integration” method.

5.5.2 Types of User Defined Content in Drupal

Once Drupal knows what possible languages to use on a site and can select the language or languages used on a page, it is important to identify the targeted objects for multilingual presentation. The list of different objects that need multilingual content support are as follows:

1. A Drupal site has various basic settings like the site name, slogan and logo image. These settings are stored as so called **variables** by Drupal.
2. User specified content is stored in objects called **nodes**. These nodes have some common base properties like the author of the content and whether it is published or not. Nodes can have any number of fields (date, location, excerpt, attachments, and so on) provided by contributed and built-in modules.
3. There can be different types of nodes, called **content types**. These content types specify some basic properties of what fields a node provides and how these are called.
4. Nodes can be categorized in a **taxonomy** system. Categories evolve around **vocabularies**, having **terms** to describe content. An example of a vocabulary for photos is *Colors*, containing terms like *red*, *green*, *yellow* and *brown*.

5. Site navigation can be built with **menus**. Drupal has built-in menu items provided by the system, and the administrators can add arbitrary menu items.

6. The layout of a web site contains content and navigation, as well as **blocks**, pointing the user to more content, showing advertisements, and so on.

7. Users can have their own **user profiles** with administrator defined fields like *Real name*, *Email address*, *Skype name* and others.

8. Drupal has an included new aggregator with RSS and Atom support that has **aggregator categories and feeds** set up on the web site.

These different types of objects need different levels of language support. Content type and user profile field names are basic string properties of objects, and therefore separate object instances are not required to translate them. Nodes, on the other hand, usually require separate instances to support workflows and adequate permissions for each content object. Taxonomies and menus are borderline, sometimes requiring different instances and sometimes forbidding different instances depending on the actual requirements imposed by the built web site. To provide a consistent translation infrastructure, my plan was to implement a translation system based on separate instances for nodes, while implementing a so called “dynamic text translation” mechanism for simpler objects.

5.5.3 Content Language

The first task in implementing content translation is to add support for content items to relate to the different languages set up on the web site. This feature already benefits those types of web sites in which multiple content items are posted in different languages, but these are not related. From single author blogs to complex community news sites, this feature is useful even without translation support. Requirements for content language support include the following:

1. If language functionality is not turned on, submitted content should be flagged as “language independent”, therefore allowing site editors to later assign languages to content.

2. If the language functionality is turned on, content types selected to have multilanguage support should provide an interface to associate a language with a content object.

5.5.4 Content Translation

To make use of many of the features already available for nodes like different workflows, permissions, and revision tracking, it is vital to have node instances as translations of other node instances, as previously discussed. This means that a node translation system similar to Plone's should be implemented. Requirements for this feature include the following:

1. Only content types with language functionality enabled should have translation support, and only if an actual node has a language associated.

2. Unlike previous implementations of the i18n and localizer module suites, revisions of nodes should form a base of translation associations so translators can be informed of outdated translations and the status of different language versions can be automatically tracked.

3. The system should allow for sharing fields between different node instances.

This allows different language versions to have their own web addresses, and it supports import and export features and news feeds that are generated by the system.

5.5.5 Dynamic Text Translation

Replacing certain properties of an object is a viable way to translate it to a different language when its translations are not required to have revision support, multiple authors or permissions. User profile forms defined by the administrator have such properties, as do site settings and other simple objects defined on the web site.

To translate such objects, the system first needs to know about the properties it will translate so it can distinguish between the ones it needs to store and look up translations for and the ones it needs to leave unmodified. This requires meta information about these objects defined in Drupal. Implementing the XML descriptor approach, as utilized by Joomla, does not meet our needs because a list of properties is not enough to present a user interface, as shown on the Joomla screenshots. Information about the user interface is also required to show the user who will edit the translations.

It is important to investigate ways to integrate translations of different texts in Drupal. Content translation and interface translation already have different approaches. Dynamic text translation of simpler objects should not provide a third confusing way, but rather integrate to the interface translation framework.

The tasks for this feature are the following:

1. Enable Drupal modules to provide meta information about objects defined in the module and their properties to be translated.

2. Allow the import and export of translations of objects for interaction with external services or agencies.

3. Provide context sensitive form elements and help text dependent on the type of value being edited.

5.6 Translation Workflow

Drupal includes very simple workflow features by default. Content objects can have flags like: “published”, “displayed on the front page”, “with change management support,” etc. These flags have default values for every content type and are editable only by site administrators or content editors. This means that if a content item is created by someone not privileged to modify these flags, the so-called “default workflow” will be started with the content object. Later on, editors can modify flags on the content object any way they like.

Most complex web projects require tracking of documents’ status, as well as running several actions when documents change status. For this reason, Drupal has a workflow and an actions module, which jointly allow web site implementors to define possible states and transitions and assign actions to the transitions.

For example, a document can be tracked through a “draft”, “editing”, “accepted”, “to translation” and “published” cycle. Actions can be set to inform translators when a document reaches the “to translation” state, or automatically set the “published” flag of a content object when the “published” state is reached.

5.6.1 Limiting Permissions Based on Workflow

Drupal provides role-based permissions by default, that allow users to possibly have an editorial role to edit all properties of all content objects. Although support for content level permissions is built into the system, no user interface is provided for configuring it by default. Contributed functionality can hook into the system to provide a means to set content level permissions because different use cases require specific user interfaces, rules and automation to set up content level permissions.

Permission levels should be restricted to support a translation workflow. While a document is in the “draft” stage, both the content author and editor should be able to modify it. After a document is “accepted” for publication and is sent out “to translation”, modification by the author should not be allowed. This is a simple example of a translation workflow, but actual cases might involve different requirements, so a general solution was planned.

I decided to tie user roles to workflow states and provide permission settings based on these three dimensions. This way when a node moves from “draft” to “accepted”, the editing privileges can be revoked from the author in the use case above. At the same time, translators should have privileges to view content objects in the “to translation” stage, even if those are not yet published to the web site. The planned system should be general enough to support any content permission changes throughout workflows configured on a convenient web interface.

5.6.2 CAT Based Workflows

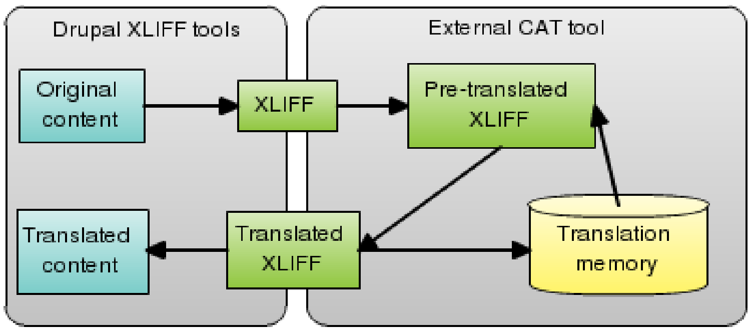


Figure 5.4: Maximalist XLIFF transformation method planned for Drupal

As described earlier, most professional translators use their dedicated tools supporting storage and reuse of previous translations, so an interface with these tools is a requirement if large volumes of to-be-translated content are handled by the system, or if in-house translation know-how is not present.

A Computer Aided Translation workflow support tool was not present for Drupal at the time I reviewed the system for my thesis, so I decided to implement interfaces to import and export in the XLIFF format as a contributed module. Requirements for this functionality include those listed below:

1. Work with the content types shipped with Drupal by default.
2. Enable administrators to limit access to the functionality based on permissions and workflow states.

3. Provide an API to reuse the XLIFF import and export functionality by possible workflow actions (eg. sending a mail with an XLIFF document to the translator when a content object gets ready for translation).

Chapter 6

Implementing a Solution with Drupal

6.1 Source Code Based Interface Translation

6.1.1 Installer Localization Support

Drupal's interface translation API is one of the most invoked set of functions, so extending the internals of the implementation would hurt runtime performance of Drupal sites, which is not acceptable. So the first requirement of this functionality was slightly modified: the interfaces provided by the translation API, but not the API itself should be reused. The new install-time interface translation functionality works with a memory backend instead of the database. Standard locations were defined for the install profiles' translation files, so all translations found by the installer for a specific profile are offered to the user. Once a language is selected, the relevant Gettext PO file is loaded into memory and these strings are never imported to the database. The work I have done to implement this functionality includes the following:

1. Add language selection screen to the Drupal installer to allow the administrator to select a language from the ones available for the current profile.

2. Allow for in-memory storage of translation strings while maintaining the language used throughout multiple HTTP requests required to complete the installation.

3. Modify the translation template extractor to generate a separate installer template, therefore recognizing the text used in install time.

4. Test the Hungarian translation and fix some problems I encountered in the installer itself.

Drupal 5.0 shipped with my improvements and the translation packaging scripts were modified by Derek Wright to include the installer translations in downloadable packages separately.

6.1.2 More Efficient Translation Packaging and Importing

Requirements specification was late for this feature to get into Drupal 5.0, so I started development as part of the new “autolocale” contributed module and “localized” install profile [40]. The “localized” profile adds autolocale module to the enabled modules list, so that when the installation process is done the initial translation import can be performed. Translation files are searched in standard places under module directories. Jakub Suchy implemented the functionality of monitoring module changes, so when a new module is enabled, translations for that module are imported.

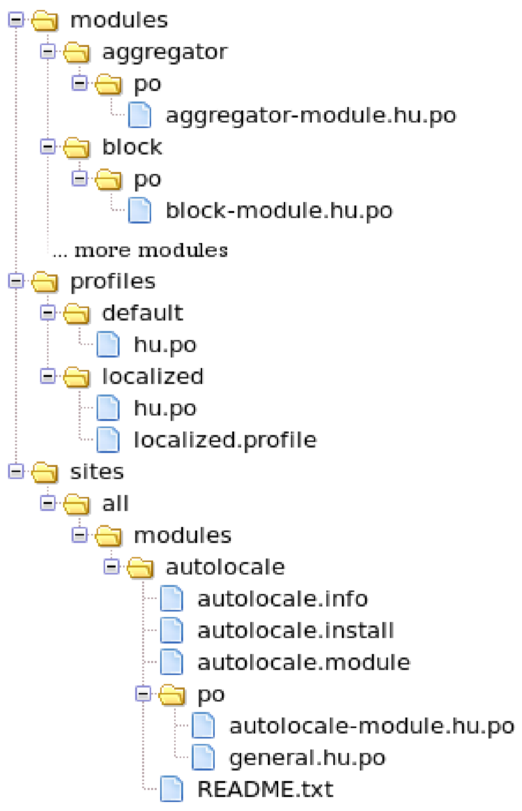


Figure 6.1: The structure of the Hungarian translation package

To have separate translation files in standard locations, I have developed new packaging scripts that generate a package structure similar to Drupal 5's own directory tree. Once a translation package is uncompressed, the translation files, autolocale module and localized profile files fall into place where Drupal expects them. I have run a beta test on my new packages with the Hungarian Drupal community and the results showed that most of the requirements were met.

Unfortunately, two requirements were not possible to meet in Drupal 5.x for architectural reasons. Due to the Gettext based translation system, Drupal only knows about string pairs, and there is no exact information maintained about where those strings are actually used in the source code. Therefore it is not possible to remove unused strings when a module is turned off. It is unlikely that this will get fixed in Drupal 6.

The other limitation we faced was in the handling of the installer process and form submissions. We could only meet the requirement of performing translation imports under limited resources by dividing the import process to multiple HTTP requests. Unfortunately, Drupal 5 does not offer a way to perform multiple "batched" HTTP requests in the installer process or on form submissions.

I worked closely with Yves Chedemois to solve this issue in Drupal 6, and as a result, Drupal 6 includes a batch API that allows progressive operations to be executed in multiple HTTP requests. I also submitted a polished version of the install profile and the developed import functionality, and got it included in Drupal 6.

The work I have done to implement this functionality includes the following:

1. Implement a new autolocale module and a localizer install profile for Drupal 5. Adapt it to the batch API and update the functionality to be included in Drupal 6.

2. Define a new packaging format for translations and implement a packaging tool to generate packages in this format.

3. Conduct initial testing on the Hungarian translation and public testing within the community.

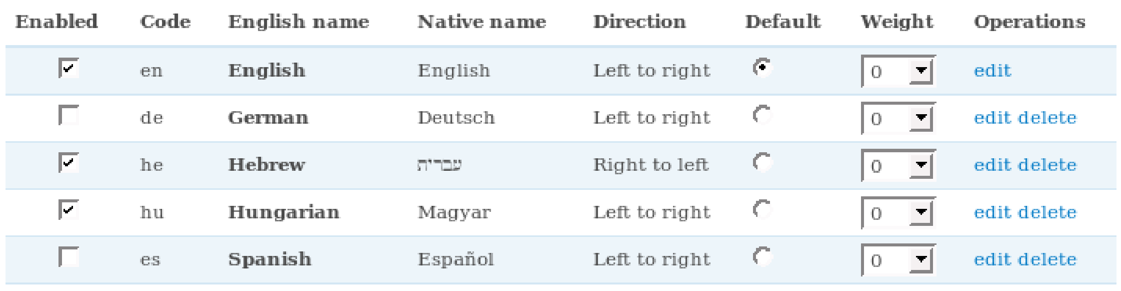
6.2 Language Management Functionality

I collaborated with Jose A. Reyero to implement this functionality. A new language list handler was developed to support the following properties of all languages:

- An enabled/disabled flag to set when a language is active on a web site.

- An RFC 4646 language code to replace the deprecated ISO 639 code used earlier.
- A language name in English, used for translation to display names in the appropriate interface language.
- A name in the native language that can be used to guide users to select their language, even if not knowledgeable in the current language displayed.
- Direction information with left-to-right and right-to-left support.
- Weight information that can be used to display lists of languages.
- Custom path prefix support (eg. `http://example.com/deutsch/` for German and `http://example.com/magyar/` Hungarian).
- Arbitrary custom domain support (eg. `http://example.de/` and `http://example.hu/` for the corresponding languages).

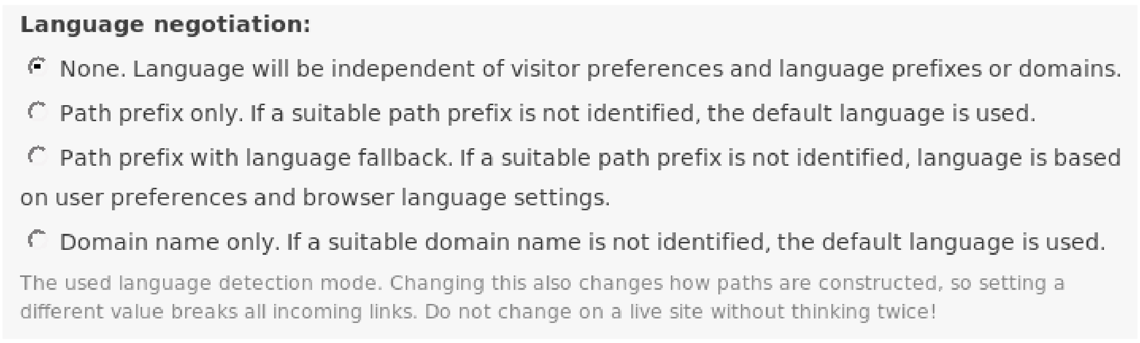
A new language management screen was implemented to provide an overview that allows toggling of the most important properties and guides the administrator to detailed editing screens for each language. I modeled this user interface after the module listing and block editing interfaces of Drupal so that the concepts applied here should be familiar to existing Drupal users.



Enabled	Code	English name	Native name	Direction	Default	Weight	Operations
✓	en	**English**	English	Left to right	●	0	edit
	de	**German**	Deutsch	Left to right	○	0	edit delete
✓	he	**Hebrew**	עברית	Right to left	○	0	edit delete
✓	hu	**Hungarian**	Magyar	Left to right	○	0	edit delete
	es	**Spanish**	Español	Left to right	○	0	edit delete

Figure 6.2: Language management interface in Drupal 6

The required process of language selection differs from web site to web site, so a sensible default cannot be specified. To ease method selection, we implemented combinations of path and domain based lookups with fallbacks, as users specified in their use cases in my research [19]. Our implementation of browser language detection and different negotiation methods got accepted into Drupal 6.



Language negotiation:

[✓ None. Language will be independent of visitor preferences and language prefixes or domains.]

Path prefix only. If a suitable path prefix is not identified, the default language is used.

Path prefix with language fallback. If a suitable path prefix is not identified, language is based on user preferences and browser language settings.

Domain name only. If a suitable domain name is not identified, the default language is used.

The used language detection mode. Changing this also changes how paths are constructed, so setting a different value breaks all incoming links. Do not change on a live site without thinking twice!

Figure 6.3: Language negotiation options in Drupal 6

6.3 User Specified Content Translation

As outlined in the requirements, two different translation methods were implemented for content translation: one for nodes based on language properties and translation association and the other for textual properties of simpler objects based solely on their textual values. I worked with Jose A. Reyero to implement solutions based on my list of requirements.

6.3.1 Content Language

A straightforward solution to assign language information to content objects will be introduced in Drupal 6. The system allows for language-less (so called "language neutral") nodes as well as nodes associated with exactly one language based on their content.

Until the multilanguage subsystem is turned on, content objects are saved as being language neutral. The multilanguage subsystem, however, allows for enabling language support on content types, thus providing a user with a selection widget to specify the language the content is being saved in.

It is possible to have content types without associated language controls, in which case new nodes created in this type are saved with the default language. For example, this allows for web site forums to always have content saved in their default language.

The implementation also allows extension functionality to hook in and alter the behavior of the language selection widget. This is important for the content translation implementation, but also opens the door for other types of automated language association tools to help users.

6.3.2 Content Translation

Although there was discussion about a generic node object relation system in the community while I was preparing the implemention of content translation, it did not materialize into a system that I would be able to build on. So I opted for a straightforward relation model instead.

Translations of the same content are organized into so called “translation sets,” which have a “source node”. The source node serves as a base to compare translations to so that the status of different language versions can be compared to the initial content. Translation sets are always created the first time a node gets a translation and make the original content object the source node. This design allows the system to track the revision identifiers of the set member nodes as updates are made to their contents. It also enables extensions that give users finer controls around the edits made to particular content. This way the revision relations can be accurately maintained.

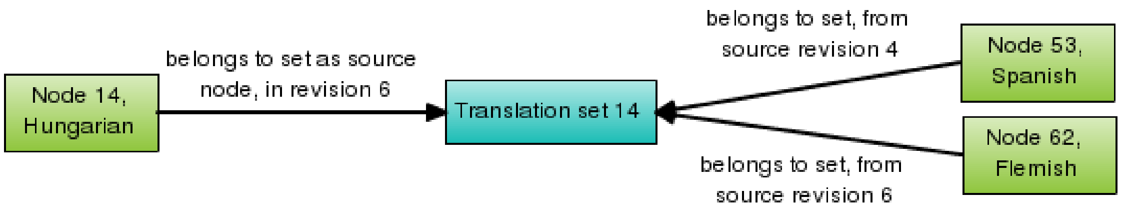

belongs to set, from source revision 4
Node 53, Spanish
belongs to set as source node, in revision 6
Node 14, Hungarian
Translation set 14
Node 62, Flemish
belongs to set, from source revision 6

Figure 6.4: Translation sets with revision tracking

The source node concept also allows for the possibility of sharing particular fields between the original node and the translation objects. Because the sharing feature is not targeted at Drupal 6, the design leaves this possibility open for contributed modules to build on. Source nodes fulfill the role of the analogous canonical content object concept from Plone, and thus are able to provide default values when properties are not provided in translations. Since Drupal 6 does not offer the possibility of stand alone properties and a complex extension, (the Content Construction Kit [41]) is required to build content types with stand alone fields that allow for the possibility of sharing fields, my design cannot go that far in itself.

6.3.3 Dynamic Text Translation

Close examination of the existing localization (locale) module in Drupal showed that it would be possible to reuse the functionality provided by that module for object translation.

The locale module stores source text and corresponding translations in different languages. This system has some assumptions about the text it is working with:

1. The localization functionality is built to translate from English to any other language. Source text is always assumed to be in English.
2. The locale module itself treats every translated string as standalone text that is not related to other strings or places where it was defined. Although the web address of the page where the string first occurred or the source file name and line number are collected, these only serve as informational helpers for the translator.
3. The module provides generic text fields as a translation interface for all texts translated through its web interface. This did not fit the requirements.
4. The locale module either works with previously imported translations or collects new untranslated source strings to translate as the user browse web pages generated by the system.

Although the locale module has good import and export functionality and a useful storage backend for translations, the above limitations needed to be lifted to be able to use it as a backend and frontend for translation.

To support different *translation domains* on top of the built-in interface translation feature, I decided to extend the module to allow for more domains defined by modules. These domains allow the implementation to distinguish between the built-in interface translation and translations of content type properties, user profile field definitions, aggregator categories and feeds, and more; each of them being in different domains. For usability reasons and to avoid ambiguity with language dependent internet domain names, this feature got the *text groups* name and was accepted into Drupal 6.

To reflect the structure of the objects in translation the modules needed to be able to provide meta information about their objects. To do this I introduced the new *locale hook*, which can be invoked in any module that defines text groups in order to get information about the defined text groups as well as the objects to be translated in those groups. Objects have their identifier property provided as well as the names of properties to translate. This way the locale module knows what source text it should look up for translations and can store the translations when asked to.

Nearly all the objects involved in dynamic text translation are defined by the administrator or site editor before being used so modules can ask the locale module to save the source text of properties. The only exception is *site settings*, which has defaults in the system if they are not manually defined. I proposed a system to designate a central place to provide site settings defaults, which is still being discussed within the developer community as of this writing.

I implemented the above object translation scheme for the aggregator and user profile modules and presented them to the community for inclusion [42].

Storage of the structure of objects presented for localization is solved by reusing the *location* field provided by the existing locale module, but which previously was only used for informational purposes. My code stores an “object property path” there that contains the object type, identifier and the property to translate. Source and translation strings can store their values just as the locale module does.

Although this solution requires locations to be strictly maintained, it makes reusing the import and export functionality seamless, which I provided and had included in Drupal 6. Having Gettext based interfacing for dynamic text translation allows users to tie into existing translation tools and provide their text to translation agencies. Because most of the text to be translated consists of a few words, or at most two to three sentences, an XLIFF based interface would be overkill.

The text groups design and the object meta information definitions allow the locale module to generate forms with information about the exact property being edited. Although the designed system has no knowledge about the required form controls for certain properties, the forms contain identifiers so Drupal’s built-in capability to alter forms presented to users allows any module to replace the generic text editing fields provided by the locale module by default.

6.4 Translation Workflow

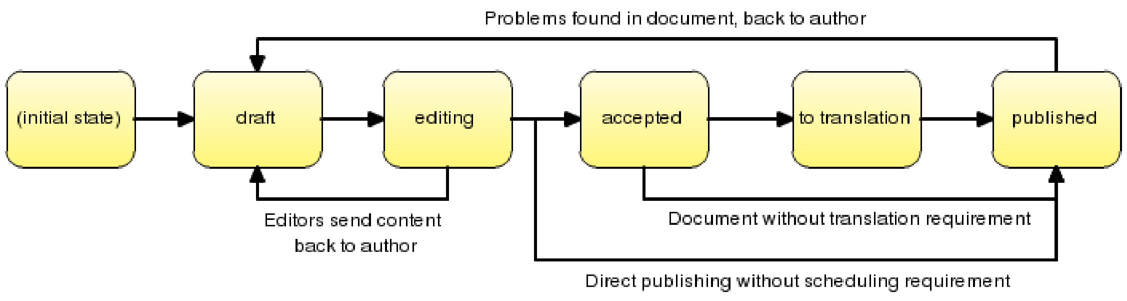

Problems found in document, back to author
(initial state)
draft
editing
accepted
to translation
published
Editors send content back to author
Document without translation requirement
Direct publishing without scheduling requirement

Figure 6.5: Sample translation workflow implemented

Using the workflow module, I have implemented the above workflow containing the states outlined in the requirements section. A sample document has an author, starting off with the content in the “draft” stage. Once she feels the content is ready for editing, she can move the document to the “editing” stage where editors can look at the content and either bounce it back to the author or advance it to “accepted” or “published” directly.

The “accepted” state is needed if scheduling requirements are in place for the document. A press release or an article for a magazine will not get published until a certain set date. Another example presented earlier in my thesis is the case of legal documents required to be published at the same time in multiple languages. Until the original document gets into the “to translation” state and all the translations get to an “accepted” state, publication of the documents should not be possible.

While it is possible to implement the above graph with the workflow module, functionality provided by the actions module is required to send emails to editors and translators when documents enter a stage relevant for them. Automatically publishing a document when it gets to the “published” state is also implemented within the actions module. I still needed to look into implementing solutions for fine grained permissions and translation specific functionality, as described in the following sections.

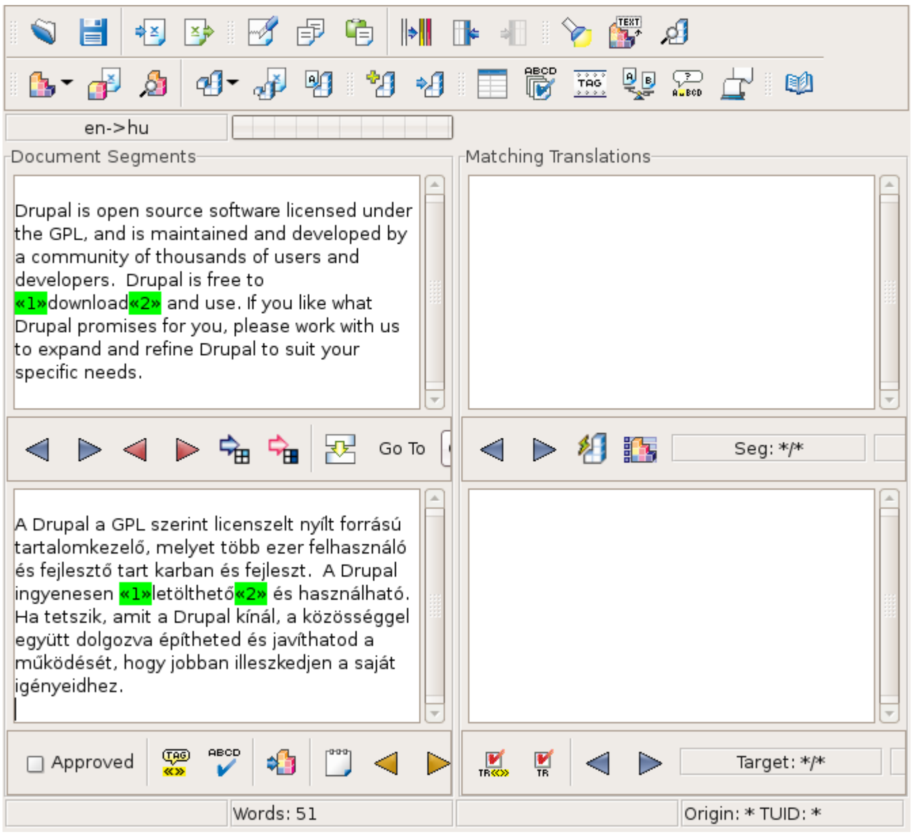
6.4.1 Limiting Permissions Based on Workflow

As outlined in the requirements, different workflow states may require different permissions for the user roles working with documents. Therefore a three dimensional permission configuration tool was implemented.

While searching for prior work, I found the workflow_access module developed by Earl Miles and discontinued a year ago. Although Drupal has changed a lot since that time, it was possible to update the module as well as fix some issues. I handed over the module to Mark Fredrickson (the workflow project maintainer), as he sees the future of such functionality close to the workflow module itself. My implementation was released as part of the workflow module suite in it’s version 5.x-1.1 [43].

6.4.2 CAT Based Workflows

While searching for some previous examples of HTML to XLIFF and XLIFF to HTML conversion, I found XSLT sheets developed by Bryan Schnabel [44] available under the GNU GPL licence, which was a perfect fit for Drupal. Based on the XSLT sheets, I was able to implement a conversion tool which adapts Drupal to a Computer Aided Translation based workflow and provides export and import functionality in the industry standard



Drupal is open source software licensed under the GPL, and is maintained and developed by a community of thousands of users and developers. Drupal is free to «1»download«2» and use. If you like what Drupal promises for you, please work with us to expand and refine Drupal to suit your specific needs.

A Drupal a GPL szerint licenszelt nyílt forrású tartalomkezelő, melyet több ezer felhasználó és fejlesztő tart karban és fejleszt. A Drupal ingyenesen «1»letölthető«2» és használható. Ha tetszik, amit a Drupal kínál, a közösséggel együtt dolgozva építheted és javíthatod a működését, hogy jobban illeszkedjen a saját igényeidhez.

Figure 6.6: Exported XLIFF editing in Heartsome XLIFF Translation Editor

XLIFF format. To allow for the largest flexibility and given that Drupal content objects usually contain a small amount of formatting to export into a skeleton, a maximalist extraction method was implemented.

The XLIFF tools extension [45] handles simple content types with a title and a body and maps the title to the HTML page title and the body to the HTML page body. The conversion component identifies document segments to generate an XLIFF document for download. The uploaded XLIFF document is assembled back to an HTML structure, from which the title and body are extracted for submission into the Drupal content database.

I have implemented a configurable workflow action to allow disabling and enabling of XLIFF import and export functionality at any transition of a complex workflow. I have tested my implementation with a sample workflow, enabling XLIFF import only when having a document in the draft state and allowing export only when in a “to translator” state.

6.5 Evaluation

Through working on my thesis, I was able to present plans and implementations on the areas defined in the previous chapter. Most of my results are already available in the development version of Drupal 6, while others are hosted as contributed modules that are downloadable separately to extend base functionality.

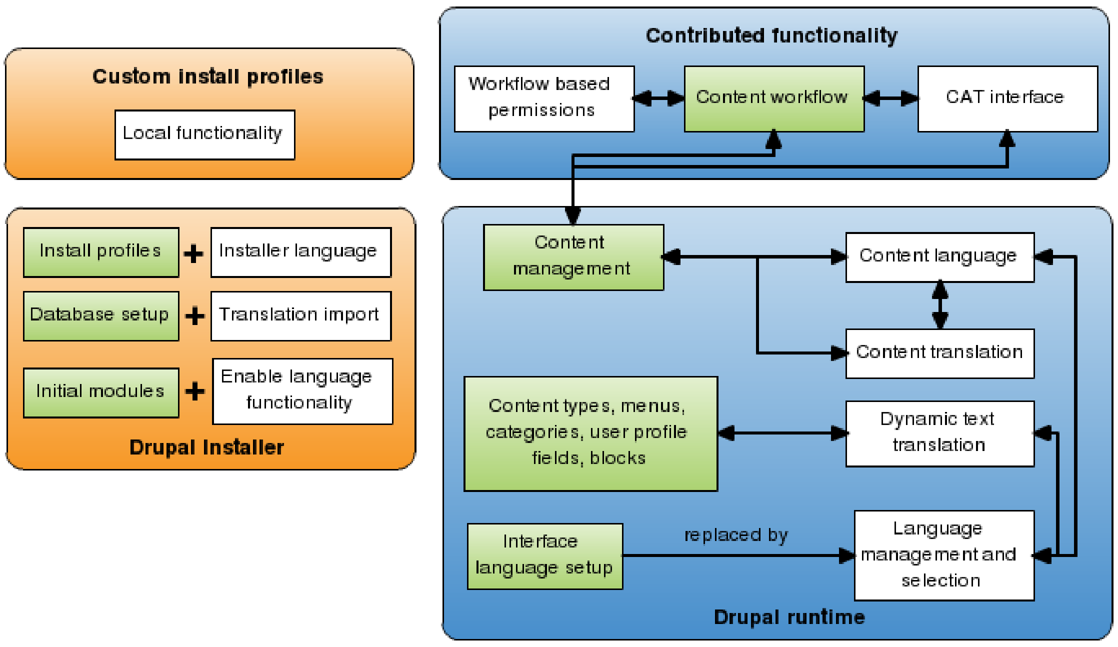


Figure 6.7: Recap of the planned Drupal language architecture

Installer language Drupal 5 included my improvements to have language selection in the installer. The translation template generation scripts I extended to support generation of installer templates were used in the community in practice.

Translation import I developed the autolocale module [40] for Drupal 5, and brought its functionality to the Drupal 6 base system with Yves Chedemois and Jakub Suchy.

Enable language functionality Drupal 5 is shipped with features I developed to detect when language functionality is required and initialize it for web site administrators.

Local functionality By implementing a custom install profile, I distributed the Hungarian interface translation with local functionality for Hungarian date format support and features for common hosts, beginning with Drupal 5.1 [38].

Language management and selection Developed with Jose A. Reyero, this new feature is included in the development version of Drupal 6 and fulfills all requirements set in the plans.

Dynamic text translation Backend modifications to support different "text groups" are included in the Drupal 6 development version. The actual object translation API is implemented in the open development environment sponsored by Development Seed [46], and the feature is proposed for inclusion in Drupal 6. Due to limited reviewer capacity and the extensive changes proposed, the acceptance of this feature is not yet known. Requirements to provide custom user interface controls for specific properties of objects are not implemented, but the design is open for pluggable user interface providers.

Content language The content language improvements developed with Jose A. Reyero are already included in the development version of Drupal 6. All requirements were met, in fact additional features are included to provide better overview possibilities of content languages on the administration interface.

Content translation Implementation of the requirements set out for content translation is ready in the open development environment sponsored by Development Seed [46], and the feature is proposed for inclusion in Drupal 6. Fortunately (unlike the dynamic text translation), the implementation of the language technologies allow this feature to live on as a contributed module because it does not require system level changes to function. The acceptance of this feature into Drupal 6 is not yet known.

Workflow based permissions I contributed the workflow-based access-control implementation to the workflow module maintainer, and it was released in Workflow 5.x-1.1 [43]. It allows for state based permission setting, which implements solutions for the requirements set out in my plans.

CAT interface I implemented and released the XLIFF Tools module [45] for Drupal 5 that allows interaction with computer aided translation tools, and plugs into workflows by restricting functionality to what makes sense in the implemented workflow.

Through working in these areas, I presented source code patches to change and expand on several aspects of how Drupal works. Provided annotated screenshots to illustrate my changes as well as created video demonstrations of the process of how specific features work and impact Drupal site builders. Some examples:

- Annotated screenshots of the text translation interface: `http://groups.drupal.org/files/localecritic.jpg` from `http://groups.drupal.org/node/3916`
- Screenshot montage of node language support: `http://drupal.org/files/issues/nodelanguage.png` from `http://drupal.org/node/137376`
- Video demonstration of install time interface translation import: `http://hojtsy.hu/drop/DrupalInstaller.avi` from `http://drupal.org/node/141637`
- Video demonstration of dynamic text translation: `http://hojtsy.hu/drop/dtvideo.mpeg` from `http://drupal.org/node/139970`

Chapter 7

Summary, Future Directions

In my thesis I have examined the special requirements of multilanguage web sites and presented technical and cultural difficulties and some existing best practices to the outlined problems. Then I presented some content management systems used in multilanguage scenarios with different project requirements, selecting Drupal for my implementation. Finally I documented plans derived from the requirements identified earlier and presented implementations for the specific needs.

Multilanguage web projects provide a very wide range of possible requirements. While I stuck to several key areas in my thesis, actual implementations might involve more specific needs, for which I tried to provide some starting points. Although the open source content management system market is very crowded, I also needed to restrict my scope by selecting a few of the available solutions to examine deeply, therefore representing different approaches to multilanguage support.

Most of the improvements that I have planned and implemented with some members in the community are now part of the upcoming Drupal 6 release, while some of them are released as contributed modules to extend the currently stable Drupal 5 system.

Due to the positioning of the release cycle and the limited reviewer capacity of the Drupal community, not all of the improvements implemented for the system have been included in the base runtime, as of this writing. The Drupal 6 code base is still open for new features and fixes, so further development will continue after I submit my thesis.

Different multilingual projects have very different needs, but there are some general directions of improvement where my work has potential to grow. Shared content properties between translations of the same content object and more focused user interfaces for online translation are areas in which future Drupal versions should provide solutions.

The improvements I presented will allow Drupal 6 users to set up web sites in languages with right to left written scripts (such as Arabic), assign languages to their content and translate documents on the web site. Workflow based permissions and computer aided translation interfacing tools will enable them to utilize Drupal in a professional multilanguage publishing scenario. The existing i18n and localizer module suite maintainers are already committed to porting their extensions and solutions to serve even wider needs on top of the new Drupal 6 architecture.

While working on my thesis, Google announced its Google Summer of Code program [47] for 2007, in which the Drupal development community is participating as a mentoring organization. I submitted an application [48] to further improve localization tools used by translators of the system worldwide and was accepted along the 20 other students under the Drupal umbrella. This means that I will further improve the multilanguage toolset around Drupal by working on project management support for translation teams and a community-oriented web based translator interface to hide the difficulties of version control systems and GNU Gettext based translations.

Given that programmatically extensive changes and new programming interfaces have been introduced into the system in combination with a simple-to-use web based interface, I hope that more developers will implement language tools. I also hope that it will help them to be aware of multilanguage issues so they develop their extension modules with internationalization in mind, further advancing the set of solutions for web site implementors.

Acknowledgements

- Thanks to my consultant, Péter Hanák, for his guidance and direct suggestions throughout my work. My thesis would not have been this complete without the reviewer tips provided by Dries Buytaert, Eric Gundersen and Péter Adamkó.
- I would like to thank Development Seed for infrastructure support for the internationalization work done for Drupal 6. Their subversion repository and mailing list setup provided a discussion forum and a temporary Drupal fork to work on safely and in the open, while Drupal development progressed.
- Thanks to the Hungarian Drupal user community for testing my new packaging format and import solutions for interface translations and providing valuable feedback on how can it be improved. Jakub Suchy did a great job as an early adopter within the Czech community and provided fixes and improvements for the code.
- I had the pleasure to collaborate with Jose A. Reyero, Károly Négyesi, Dries Buytaert, Steven Wittens and Doug Green on the language management functionality for Drupal 6. Content and dynamic text translation functionality was developed in collaboration with Jose A. Reyero.
- And last but not least, I would like to thank Bonnie Bogle for reading and copy editing my thesis.

Glossary

API Application Program Interface. A set of calling conventions that allow access to specific services.

ASCII American Standard Code for Information Interchange. A character encoding based on the English alphabet.

CAT Computer Aided Translation. A process of reusing existing translation memories.

CSS Cascading Style Sheets. Allows for specification of presentational information, mostly for web pages.

CMF Content Management Framework. A set of programs and APIs used to build customized content management solutions.

CMS Content Management System. A set of programs used to manage web site content, users, presentation, and more.

FTP File Transfer Protocol. A plain text based protocol used to transfer files.

HTML Hypertext Markup Language. A language used to produce web pages.

HTTP Hypertext Transfer Protocol. The base protocol of all communication on the web.

ICANN Internet Corporation for Assigned Names and Numbers. Responsible for the global coordination of the internet's system of unique identifiers.

IDNA Internationalized Domain Names in Applications. A domain name format that allows for Unicode character representation.

IRI Internationalized Resource Identifier. Extension of the URI concept with support for Unicode characters.

ITS	Internationalization Tag Set. A W3C recommendation that specifies common tags for marking up XML documents for localization.
LISA	The Localization Industry Standards Association. Group of experts working on localization standards.
LTR	Left to right. A script of a language in which text is written from left to right, such as the Latin script.
MIME	Multipurpose Internet Mail Extensions. Defines a possibly multipart message structure capable of having non US-ASCII headers and content.
MO	Gettext Machine Object. A binary representation of the PO format.
PO	Gettext Portable Object. Text file format used to exchange interface translations.
POT	Gettext Portable Object Template. Text file format used to provide usually English source strings for interface translation.
RFC	Request For Comments. A series of memoranda encompassing new research, innovations, and methodologies applicable to internet technologies.
RSS	Really Simple Syndication. An XML based format used to share content items between web sites and applications.
RTL	Right to left. A script of a language in which text is written from right to left, such as the Arabic script.
TLD	Top Level Domain Name. The last part of domain names, handled by the root name servers.
TMX	Translation Memory eXchange. A format used to exchange and store existing translation information.
URI	Uniform Resource Identifier. Identifier (typically an address) of a resource (eg. a web page). Used as a synonym to URL in this thesis, although generally is a broader term.
URL	Uniform Resource Locator. A subset of URIs specifying the location of a resource.
UTF	Unicode Transformation Format. A character encoding mapping method for Unicode characters.

W3C World Wide Web Consortium. Develops specifications, guidelines, software, and tools to be used on the web.

WCMS See CMS.

WebDAV Web-based Distributed Authoring and Versioning. Aims to make the WWW a readable and writable medium.

XLIFF XML Localization Interchange File Format. An industry standard used for translation information interchange.

XSLT Extensible Style Language Transformation. An XML based declarative transformation language to transform XML documents into other XML documents.

List of Figures

List of Tables

Bibliography

[1] *Portal do Governo Brasileiro*, `http://www.brasil.gov.br/`.

[2] *W3C Internationalization (I18n) Activity*, `http://www.w3.org/International/`.

[3] *Practical & Cultural Issues in Designing International Web Sites*, Richard Ishida, Fundamentos Web 2006 Conference, Oviedo, Asturias, Spain, 3 October 2006, `http://www.w3.org/2006/Talks/fundamentos-web-ri/`.

[4] *Uniform Resource Identifier (URI): Generic Syntax RFC*, January 2005, `ftp://ftp.rfc-editor.org/in-notes/rfc3986.txt`.

[5] *Internationalized Resource Identifiers (IRIs)*, January 2005, `ftp://ftp.rfc-editor.org/in-notes/rfc3987.txt`.

[6] *An Introduction to Multilingual Web Addresses*, 2 December 2006, `http://www.w3.org/International/articles/idn-and-iri/`.

[7] *ICANN Successfully Conducts Laboratory Tests of Internationalised Domain Names*, 7 March 2007, `http://www.icann.org/announcements/announcement-4-07mar07.htm`.

[8] *Hypertext Transfer Protocol – HTTP/1.1 RFC*, June 1999, `http://www.rfc-editor.org/rfc/rfc2616.txt`.

[9] *Multipurpose Internet Mail Extensions (MIME) RFC*, November 1996, `ftp://ftp.rfc-editor.org/in-notes/rfc2045.txt`.

[10] *Unicode Home Page*, `http://www.unicode.org/`.

[11] *Character sets & encodings in XHTML, HTML and CSS*, 2 February 2006, `http://www.w3.org/International/tutorials/tutorial-char-enc/en/index.html`.

[12] *HTML 4.01 Specification*, W3C Recommendation, 24 December 1999, `http://www.w3.org/TR/html4/`.

[13] *BCP 47: Tags for Identifying Languages (RFC 4646) and Matching of Language Tags (RFC 4647)*, September 2006, `http://www.w3.org/International/core/langtags/rfc3066bis`.

[14] *Language tags in HTML and XML*, 9 November 2006, `http://www.w3.org/International/articles/language-tags/Overview.en.php`.

[15] *Cascading Style Sheets, level 2*, W3C Recommendation, 12 May 1998, `http://www.w3.org/TR/REC-CSS2/`.

[16] *Script direction and languages*, 20 November 2006, `http://www.w3.org/International/questions/qa-scripts.en.php`.

[17] *Text Expansion (or Contraction)*, `http://www.omnilingua.com/resourcecenter/textexpansion.aspx`.

[18] *Designing Web Usability: The Practice of Simplicity*, Jakob Nielsen, New Riders Publishing, Indianapolis, December 20, 1999, ISBN 1-56205-810-X, pp. 312-344.

[19] *Looking for internationalization use cases*, 4 October 2006, `http://groups.drupal.org/node/1545`.

[20] *Polish translation of Drupal's aggregator module*, `http://cvs.drupal.org/viewcvs/drupal/contributions/translations/pl/aggregator-module.po`.

[21] *GNU Gettext*, `http://www.gnu.org/software/gettext/`.

[22] *The Localization Industry Standards Association*, `http://www.lisa.org/`.

[23] *Internationalization Tag Set (ITS) Version 1.0*, 3 April 2007, `http://www.w3.org/TR/2007/REC-its-20070403/`.

[24] *Systran*, `http://www.systransoft.com/`.

[25] *SDL Trados*, `http://www.trados.com/`.

[26] *Joomla*, `http://www.joomla.org/`.

[27] *Packt Open Source Content Management System Award*, `http://www.packtpub.com/award`.

[28] *JoomFish*, http://www.joomfish.net/.

[29] *Feature planning for Joom!Fish 1.8*, 5 September 2006, http://forum.joomla.org/index.php/topic,61981.msg466467.html#msg466467.

[30] *TYPO3*, http://typo3.com/.

[31] *T3locallang Documentation*, http://typo3.org/documentation/document-library/core-documentation/doc_core_api/4.0.0/view/7/2/.

[32] *New module to facilitate translations of web sites in TYPO3*, March 23, 2007, http://www.ditnetwork.de/startseite/einzelansicht/mitglied////866cdb41e7/articel/23/106.html.

[33] *Plone*, http://plone.org/.

[34] *GNOME.org CMS selection*, October 31, 2006, http://live.gnome.org/GnomeWeb/CmsRequirements.

[35] *Drupal*, http://drupal.org/.

[36] *Multilingual content-management with LinguaPlone*, Geir Bækholt and Helge Tesdal, Plone Conference 2004, Vienna, 29 September 2004, http://plone.org/news/linguaplone-tutorial.

[37] *Pro Drupal Development*, John K. VanDyk and Matt Westgate, Apress, Berkeley, California, April 16, 2007, ISBN 1-59059-755-9.

[38] *Hungarian Drupal install profile*, http://drupal.hu/files/hu-5.1.tar.gz.

[39] *Hungarian locale functionality (mini modules)*, http://drupal.org/project/hungarian.

[40] *Auto locale import module for Drupal*, http://drupal.org/project/autolocale.

[41] *Content Construction Kit*, http://drupal.org/project/cck.

[42] *Introduce dynamic object translation API*, http://drupal.org/node/141461.

[43] *Drupal Workflow module, version 5.x-1.1*, April 26, 2007 http://drupal.org/node/139438.

[44] *xliffRoundTrip Tool*, https://sourceforge.net/projects/xliffroundtrip/.

[45] *XLIFF Tools*, `http://drupal.org/project/xliff`.

[46] *Infrastructure for the i18n group*, `http://groups.drupal.org/node/2858`.

[47] *Google Summer of Code^{TM}*, `http://code.google.com/soc/`.

[48] *Tools for Drupal translation teams and users*, `http://code.google.com/soc/drupal/appinfo.html?csaid=CCB11E7904E4B8C5`.