TAMK University of Applied Sciences
Business Information Systems
Kalle Kipinä

Final thesis

Internationalization Considerations in a Custom-built PHP Web Application

Supervisor Maritta Hoffrén, MA
Commissioned by Translatum Oy, Production Coordinator Heikki Hakala
06/2009

TAMK University of Applied Sciences
Business Information Systems

Author	Kalle Kipinä
Title	Internationalization Considerations in a Custom-built PHP Web Application
Number of pages	47
Graduation time	30.6.2009
Thesis supervisor	Maritta Hoffrén, MA
Commissioned by	Translatum Oy, Production Coordinator Heikki Hakala

Abstract

The main aim of this work was to explore the concept of internationalization in a custom-built web application platform written in PHP. A secondary aim was to provide an application programming interface and a reference for the developers of the web application, as to how to make the application support localization. The interface must offer support for a JavaScript implementation in addition to the PHP implementation.

To be able to explore internationalization, the concept was first explained on a theoretical level based on literature on the subject. Before the theoretical concept was worked out, the context for the exploration was provided by describing the environment in which the web application is being developed. The exploration of internationalization itself was performed by comparing and contrasting existing solutions in various programming languages and on various levels. PHP's internal support for internationalization was thoroughly examined.

To achieve the secondary goal, it was necessary to apply the results gathered while striving for the primary one. The concept exploration on a theoretical level produced a list of functionalities needed for the web application platform to achieve internationalization. The same theoretical background could be implemented on both PHP and JavaScript.

The results of the theoretical exploration can be used on a general level when considering internationalization implementations on other application platforms. The general usability of the programming interface and its description is more limited, although the implementation can provide ideas for other platforms as well.

Some work remains to be done in validating the user input in when it comes to numbers and dates. Also, providing a solution for keeping the translatable content up-to-date will be needed. The plan described in the results of this work will help in designing the solution. Another point of interest would be to examine whether it could be possible to use a more lexically aware translation system to provide correct matching of numbers and nouns, for example. This capability could be provided by the GNU gettext system which is briefly touched upon in the theoretical portion of this work.

Keywords Internationalization, Localization, Web application, PHP, JavaScript

Tampereen ammattikorkeakoulu
Tietojenkäsittely

Kirjoittaja	Kalle Kipinä
Työn nimi	Itse kehitetyn PHP-pohjaisen web-sovelluksen kansainvälisyyden tuesta
Sivumäärä	47
Valmistumisaika	30.6.2009
Työn ohjaaja	Maritta Hoffrén, FM
Toimeksiantaja	Translatum Oy, Tuotantokoordinaattori Heikki Hakala

Tiivistelmä

Opinnäytetyön pääasiallinen tavoite oli tarkastella kansainvälisyyden tuen rakentamisen eli sovelluksen kansainvälistämisen käsitettä PHP-pohjaisessa web-sovelluksessa, joka on kehitetty ilman kolmannen osapuolen sovellusalustojen apua. Toisena tavoitteena oli kehittää ja dokumentoida ohjelmointirajapinta, jonka avulla itse kehitetty web-sovellus saavuttaa tavoitellun kansainvälisyyden tuen. Ohjelmointirajapinnan tulee toteuttaa kansainvälistämisen vaatimat toiminnallisuudet niin palvelinpuolen PHP-koodissa kuin asiakaspuolen JavaScript-koodissakin.

Käsitteen syvällisen tutkimisen mahdollisti sen teoreettisen taustan selvittely aihetta käsittelevän kirjallisuuden avulla. Taustan ymmärtäminen puolestaan oli mahdollista vasta, kun työympäristö, siis web-sovelluksen kehitysympäristö, oli esitelty. Kansainvälistämisen tutkimisen välineenä käytettiin kirjallisuuskatsausta, jossa erilaisia valmiita, eri ohjelmointikielillä kirjoitettuja kansainvälisyyden tuen toteutuksia vertailtiin eri tasoilla. PHP:n sisäinen tuki kansainvälisyysominaisuuksille selvitettiin läpikotaisin.

Toissijaista tavoitetta täytettiin soveltamalla kerätyn teoreettisen aineiston tuloksia käytäntöön. Aiheen teoreettinen tutkiminen tuotti listan toiminnallisuuksista, joita web-sovelluksen ohjelmistoalustan olisi tarjottava tukeakseen kansainvälisyyttä. Sama teoreettinen pohja oli käyttökelpoinen sekä PHP- että JavaScript-toteutuksissa.

Teoreettisen pohdinnan tulokset ovat käyttökelpoisia yleisellä tasolla, kun kehitetään ja tutkitaan uusien ohjelmistoalustojen kansainvälisyyden tukea. Ohjelmointirajapinta kuvauksineen ei ole yhtä monikäyttöinen, vaikka senkin toteutus voi toimia mallina muille alustoille.

Kansainvälisyyden tuen mahdollistava työ on edelleen kesken joiltain osin. Käyttäjien syötteen tarkistamisessa on kehittämisen varaa, samoin kuin käännettävän sisällön päivittämisen mahdollistavassa järjestelmässä. Työn tuloksissa esitetty suunnitelma auttaa tämän järjestelmän suunnittelussa. Kiinnostava jatkotutkimuksen aihe voisi olla myös käännösjärjestelmän vaihtaminen sellaiseen, joka tukee kielellisesti yhtenäisempää käännöstä, esimerkiksi numeraalien ja substantiivien oikeanlaista yhdistämistä. Tällaiseen voisi kyetä opinnäytetyön teoreettisessa osassa sivuttu GNU gettext -järjestelmä.

Avainsanat Kansainvälisyyden tuki, Lokalisaatio, Web-sovellus, PHP, JavaScript

Table of Contents

1 Introduction

In this chapter I will give a description of the background of the thesis. After the background I will present a short introduction to the subject matter. I will list the goals and the overall structure of the work briefly. The background will receive a little more focus than the rest of the subjects in the chapter.

1.1 Background

Translatum Oy is a translation service provider located in Tampere, Finland. It has a branch office in Helsinki, Finland and in Paris, France and a close co-operative relationship to a company bearing the same name in Tallinn, Estonia. All the offices have been using a web-based project management solution for a number of years. The custom in-house system has been in development for such a long time that there are hardly any larger concerns with its quality any longer.

With time, the company's needs have however grown larger, and a new version of the project management application is being developed. As a part of the conceptualization plan of their product, language translations, Translatum has shifted the responsibility for the development from an external consultant to the internal IT department. The main idea of the new development project is to incorporate all the good ideas present and in use in the old system into the new system as well, as well as to modernize the development platform.

The new system is being built from scratch on the LAMP (Linux-Apache-MySQL-PHP) platform so as to provide a platform that requires no additional licensing costs, as opposed to the Windows-based platform used in the old system. A decision was made not to use any pre-existing PHP-based framework, although a number of possibilities were presented. This decision raises another need, namely to build a web application platform starting on a clean slate. The application will be enriched with some JavaScript-based functionality, creating a more responsive user experience.

One of the requirements set for the web application platform was that it should provide support for localization because, according to the plans, the new project management system would become a commercial product in its own right and this would offer one

competitive advantage among similar products. A localizable application would be more widely accepted in such markets as have less in common with an English market than the traditional Western European one.

1.2 Goals

The main aim for this work is to explore the concepts of localization and internationalization with respect to the system being built. A goal supporting this aim is, in addition to theoretically explaining the concepts, to benchmark existing solutions in internationalization. To achieve the main goal, I will also present the circumstances in the development project, including the development environment and the server software layout for the web application.

The concepts found will be applied to the application, and its developers will be offered guidelines by which they can make the application truly localizable. These guidelines will take the form of coding practices and practical documentation on the internationalization functions. Additionally, as a product of the thesis, a system will be planned and, time permitting, also developed, for maintaining the translatable strings in the application during its continued development.

1.3 Structure of the Thesis and Primary References

Following this introductory chapter, I will present the basic server software layout and its key components used in the project management system. The third chapter will present the development environment in a little more detail.

In the fourth chapter I will explore the concept of internationalization in general. The GNU website (2009) proved to be a valuable source for this exploration, as do the thoughts of Bert Esselink in his book A Practical Guide to Localization (2002). The fifth chapter is dedicated to an analysis of four different internationalization implementations. This is the main body of the theoretical handling of the work, and this is where the main sources are used as well. A very important reference used throughout the latter parts of the thesis is the documentation on the PHP Extension Community Library (PECL) extension called Intl on the php.net web site (2009).

After the analysis, in the sixth chapter, the results will be applied to the new application platform under development, resulting in documentation on the aspects concerning internationalization in the application's development procedures. In the final chapter I will present the results in a more condensed form and take a look on how the work done corresponds to the basic aims. Additionally, I will consider the future of the project in light of this work.

2 Web Applications Using the AMP Platform and Ajax

In this chapter I will present the basic platform on which the new project management system is being developed. In addition to the basic server components, I will cast a brief look into the concept of *Ajax* and how it can serve in making the user experience of any web application more fluid. As Ajax is not without its share of problems, I will also present a solution that will eliminate most of the woes concerning differences in functionality between different user platforms: the *MooTools JavaScript Framework*.

2.1 Apache, MySQL and PHP

The basic server platform chosen for the project is known by the acronym *AMP*. The acronym stands for *Apache* as the web server software, *MySQL* as the database management system for data storage, and *PHP* as the scripting language offering dynamic capabilities and the business logic layer.

The Web Server Software: Apache

The Apache HTTP Server is one of the most widely used web server software products on the Internet today (Appu 2002, 27). It is being developed by the Apache Software Foundation, under a project called Apache HTTP Server Project (Appu 2002, 28). As web servers do, it processes and sends responses to requests issued by clients over the *Hypertext Transfer Protocol* (HTTP) (Appu 2002, 3).

In addition to the basic functionality, Apache HTTP Server provides its users with a horde of configuration possibilities. These range from basic usage as a server for static web content to an advanced configuration as a single server instance serving a multitude of sites on the same server computer. The possibilities also include offering dynamic content provided by different programming languages (Appu 2002, 31-32). Apache's range of use can be widened even more by using add-ons called modules (Appu 2002, 36), which can enable for example such services as rewriting requests sent by the clients, where the rewriting is done by a module called *mod_rewrite* (Appu 2002, 356-357) or using designated directories in the home directories of the system users as

sources for the served content, a service provided by a module called *mod_homedir* (Appu 2002, 208). Even the PHP scripting language is used as an Apache module in most cases (Appu 2002, 214).

The Data Warehouse: The MySQL database management system

Developed by the MySQL AB company, the MySQL database management system (DBMS) is an open source alternative to commercial products such as *Oracle DBMS* (Sheldon & Moes 2005, 8). Its version number 5 introduced capabilities for modern solutions such as prepared statements (MySQL 5.1 Reference Manual :: 12.7 SQL Syntax for Prepared Statements 2009), serving as a way to optimize performance, and triggers and for integrity checks for example, a vital part in keeping the data consistent throughout all the tables in the database (MySQL 5.1 Reference Manual :: 19 Stored Programs and Views 2009).

The Logic behind the Scenes: the PHP Scripting Language

The PHP scripting language, whose name is, according to the Frequently Asked Questions page in the PHP web site, recursively short for PHP: Hypertext Preprocessor (PHP: General Information - Manual 2009), is today more like a programming language than purely a scripting one. Having reached version number 5, it includes such vital programming tools as built-in, native support for object-oriented programming and APIs for several different database drivers (Converse, Park & Morgan 2004, 5).

Interestingly considering the theme of this work, the language contains some tentative functionality regarding international aspects. As will be shown later in the thesis, the support of the current version for the concepts is not, however, mature enough to be used as such. Additional functions must be built in to the framework to achieve an acceptable level of service in this respect.

2.2 Adding Responsiveness to Traditional Web Applications

The traditional web application consists of requests made by users over the HTTP (or HTTPS [HTTP over SSL, a secured version of the protocol]) protocol to the server which then responds as instructed in the scripted pages, based on the variables sent in the HTTP(S) request (Appu 2002, 3). While this works in the basic scenario, there are some caveats. The way of making web site layouts without frames makes it difficult to update only a small part of a given web page in a controlled manner: a full page reload is required for the HTTP(S) request to find its way to the server and the response to be rendered on the user's end.

According to Snook, Gustafson, Langridge and Webb (2007, 3), the idea behind Ajax, originally short for Asynchronous JavaScript and XML, is to separate the request and response from page reloads. The client browser can send a request to the server and receive a response in the background, asynchronously from any page refreshes. The idea originates from Microsoft who was the first to introduce an *XMLHttpRequest* object, an ActiveX object which is responsible for offering the asynchronous request capabilities to the script. This capability is offered by the JavaScript engine in the browser, which also takes care of updating the display for the user as necessary (The XMLHttpRequest Object 2009). The W3's XMLHttpRequest web page also points out (2009) that actually the components of the object's name are misleading: for example, the XML part would imply that only XML is accepted as a response; however, all text based formats are supported.

As Snook, Gustafson, Langridge and Webb note (2007, 3), Ajax is nowadays actually an umbrella term for the actions undertaken by the web client, typically a browser, which involve rendering pages based on the asynchronous responses, cascading style sheets (CSS) and DOM Scripting, that is JavaScript. With this, problems are imminent. While the server responses are quite independent from the client software used, the JavaScript engine and page rendering implementations differ from one another (Newton 2008, xiv). This leads to incompatibility between the client side script, the response handling and the browser which might even break the functionality of the application completely in a worst-case scenario.

2.3 JavaScript Frameworks and MooTools

Newton (2008, xiv) is happy about the fact that there now are JavaScript frameworks that have been built to abstract the browser implementation from the actual production script. One of these frameworks is called *MooTools*, whose name, according to Newton (2008, xvi), stands for *My Object-Oriented Tools*. Tested to be compatible with a number of browsers (Proietti 2009a), the framework offers a completely transparent way of making all operations browser-independent. It offers a core with many useful shortcuts and additions to the JavaScript standard library, and a range of third-party and in-house plug-ins for added functionality (Newton 2008, xvi).

As MooTools is the framework of choice in the new system, one of the more interesting aspects is the very recent addition of support for locale-specific implementations for example in date parsing, as well as a built-in capability for using translated versions of the class strings in the system (Newton 2009). These functionalities are a part of the MooTools More plug-in library for now, in version 1.2.2.1 of the MooTools More plug-in repository (Proietti 2009b).

3 The Development Environment

In this chapter I will describe the environment in which the development is being undertaken. I will first present the server environment and a brief history behind the server structure. In the second subchapter I will take a look at an exemplary workstation used in development.

3.1 The Server Layout

There are two servers that are being used in the development environment. One, *server1* is performing as a development and a testing environment and is configured on a *Debian GNU/Linux 5.0* platform. The other is a version control server, *server2*, which runs the *subversion* server as well as a *trac* environment for issue tracking and documentation. The version control and trac server uses Debian GNU/Linux 4.0 as its operating system.

In practice, the server offering the web application platform also functions as a *samba* server on which a directory share has been activated. The share has been mapped as a network drive on the workstation. The shared directory is the developer's home directory in the Linux system, which enables a separate development environment for each developer.

3.2 The Development Workstation

The workstations used in development are running Windows XP Professional as their operating systems. As the product under development is a web application, a wide selection of browsers is a desirable feature. The more browsers the developers have at their disposal, the less likely it is that the application runs into compatibility problems. Windows is a popular platform whose selection on browsers is the widest, although not complete. The browsers on a typical development workstation include Microsoft Internet Explorer, Opera, Google Chrome, Apple Safari and Mozilla Firefox.

The mapped network drive mentioned in the previous subchapter works as storage for the source code and documentation created in the development process. The source

code files include PHP source code as well as JavaScript source code and SQL definitions for database structure and, where applicable, also database data. The main tool used in the development is the Eclipse IDE. The IDE can be equipped with an add-on called PHP Development Tools (PDT), whose version options include 1.0.3 and 2.0; the former is the latest version compatible with the older 3.3 version of Eclipse while the latter is compatible with Eclipse 3.4 only. Since the team ran into some difficulties with the Subversion client incorporated in Eclipse 3.4, they have to settle for the older version of Eclipse and PDT.

As illustrated below in figure 1, from a developer's point of view the environment consists of a desktop computer and the two servers. The Eclipse IDE run on the desktop computer deals with the source code files as well as with the version control system. If the version control system support should fail in the IDE, an alternative would be to use the subversion client software installed on the web/file server using a Secure Shell (SSH) terminal connection.

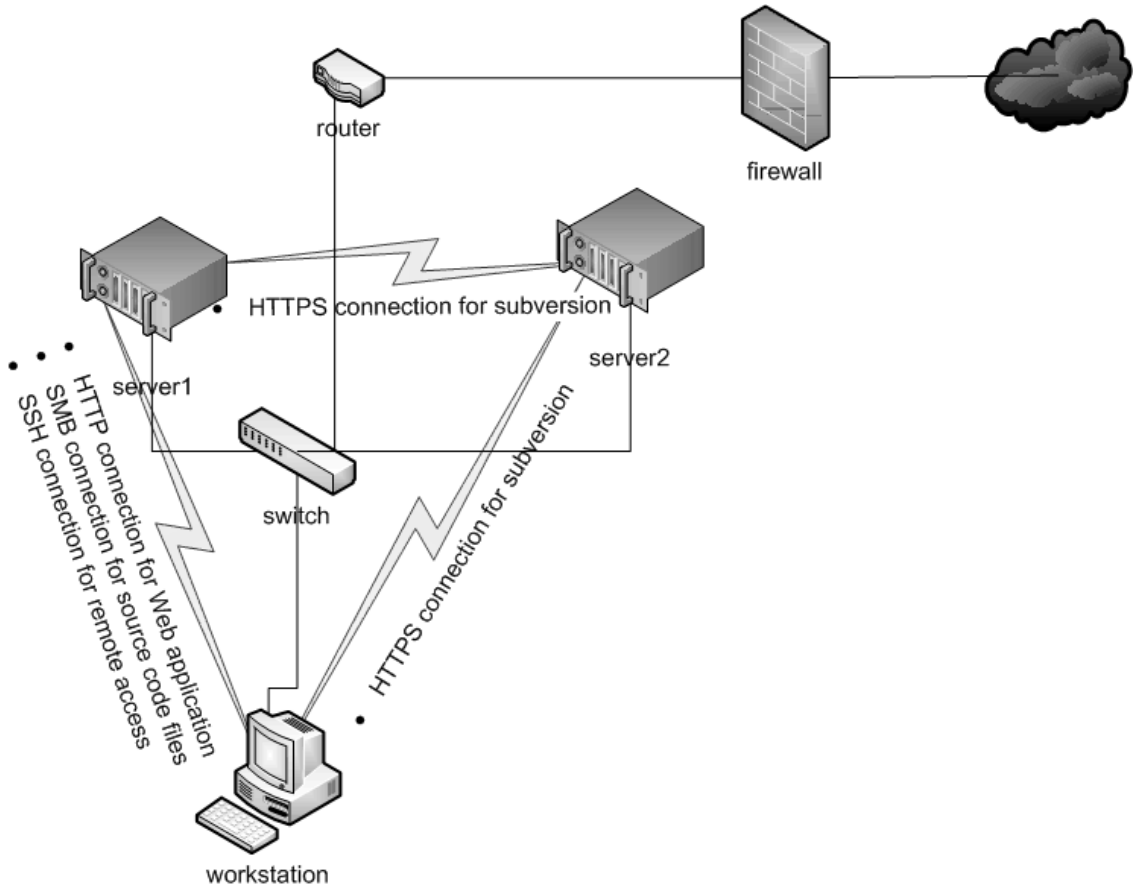

Figure 1: The development environment at Translatum

Figure 1 also shows how the entire environment is completely obscured from the internet by a firewall. The firewall is a safety measure that is surely needed but it does hinder working from home or other remote locations because there are no other means to access the internal network, such as a Virtual Private Networking (VPN) connection. If such a measure were available, the central subversion server would offer a fairly simple way of acquiring the most recent copy of the system and to continue working from anywhere.

The workstation's operating system choice is by no means locked to Windows alone. The Eclipse IDE is also available for Linux desktop environments as long as a compatible Java Runtime Environment is available. So if the developer prefers to use Linux for some reason, and does not mind a somewhat smaller selection in browsers, the development environment does not prohibit changing operating systems.

Having now described the development environment in some detail, I will next present the main concepts that this thesis deals with - internationalization, localization and the concept of locales.

4 Internationalizing and Localizing Web Applications

In this chapter I will explore approaches taken by different authors towards software *internationalization*. In addition to this, I will present the concept of *locale*. A natural conclusion to this chapter is a brief introduction to the process of *localization*.

4.1 Internationalization

Despite the similarity in terms, Localization and Internationalization are distinct concepts, one dependent on the other. As Esselink (2000, 25) points out, internationalization is required in order to make a software product localizable. Esselink further notes (2000, 26) that internationalization of a software product involves, but is not necessarily limited to, considerations regarding the correct display of accented characters; length of displayed text; date, time, number and currency formats; and cultural neutrality of icons and graphics.

Hogan (2004, 190) points out additional tasks pertinent to the internationalization considerations such as sorting and ordering of data values and setting text boundaries, both functions that are readily available in the *Java* programming language's internationalization application programming interface (API).

Parr (2006, 66-67) adds yet another aspect to the concept of internationalization. Sometimes the page design must be adapted to the needs a localized version. Examples of this are for example different character encodings or even layout changes in case a language demands different layouts for any parts of the application. In the web world, image name changes are imminent for localized images that contain embedded text.

Overall, internationalization is a concept whose purpose is to add support for different locales. What the term locale entails is explored more thoroughly in the following subchapter.

4.2 Locales

According to the GNU `gettext' utilities document (2009), a country's locale consists of not only its native language but also of a set of rules which define the local way of formatting date and time as well as representing numbers and symbolizing currency, among other things. As the document aptly puts it, "The locale represents the knowledge needed to support the country's native attributes."

GNU have put together a list of the effects a locale should have in their library implementation for the C programming language. While not directly applicable in this work since the programming language used is not C, the list does include such relevant factors as:

- Classification of which characters in the local character set are considered alphabetic, and upper- and lower-case conversion conventions
- The collating sequence for the local language and character set
- Formatting of numbers and currency amounts
- Formatting of dates and times
- What language to use for output, including error messages
- What language to use for more complex user input

(Effects of Locale - The GNU C Library 2009)

Of the listed items, the last one is significant in that none of the writers in the previous subchapter gave any significance to user input; they were all more concerned about formatting out-put.

4.3 Localization

As noted earlier, localization and internationalization are distinct concepts. They are related, however, in the sense that the latter enables the former. Whereas internationalization is performed inside the development process of the application and works as the enabler, localization can be seen as the process which takes place after, or during, the development of the application.

Yang (2007, 285) says that the process of a localization project of an application is usually started at a point where the development of the original language version is frozen, that is, when no new user interface texts are meant to appear in the application. The project kick-off is followed by a period during which the localization process is being planned and the data being prepared for different vendors. The preliminary stage is followed by the concrete localization efforts such as translation. After the work is done, testing and bug fixing take place, after which the localization process is considered completed.

On a more practical level, a list of different tools available for localization has been conceived by Esselink (2000, 359-360). He lists computer aided translation (CAT) tools and machine translation (MT) tools as the main categories. The distinction between the two is that whereas CAT tools are meant to offer help to the localization vendor, MT tools are supposed to be helped by humans and to do the main bulk of the work by themselves, automatically. Esselink further notes that while machine translation is not a widely recognized field in the software industry, there is a multitude of tools available for computer aided translation.

Computer aided translation tools can be further divided into two subgroups (Esselink 2000, 360). One is the group formed by bundles of translation memory systems and terminology tools; the other is the group of software localization tools. The former are usually used for documentation localization, including HTML text. The latter are used for simultaneous testing and localization of software products and their user interfaces. The user interfaces include such components as menus, messages and dialog boxes.

In the case of this work, choosing to use a software localization tool would be the natural choice. However, this writer has come across no tools that would be directed at PHP-based web applications. Besides, the kind of menus and dialog boxes referred to by Esselink are not parts of traditional web applications; these components are rendered, and their internationalization handled, by the web browser the user chooses to use. Hence, the choice is made easy: for the purposes of this work, the localization tools will need to consist of translation memory and terminology tools.

Having laid out the ground work for internationalization and localization concepts, it is time to shift attention to what the industry already has to offer. There are plenty of

internationalization implementations available, even in open source, but as we will see, none of these is directly suitable, or adequate, for the project management software product that is the subject of this work.

5 Existing Solutions in Internationalization

In this chapter I will describe how internationalization has been handled in some existing web application frameworks. I will try to find if a suitably mature solution for internationalization could be already built in the PHP language itself, thereby eliminating any further needs for providing a separate internationalization solution. Before that, however, I will take a look at some competing solutions to see what I should look for in PHP.

In this work, more focus is put on the internationalization of the actual software product than the localization process itself. As I will show later, some attention is going to be paid on the actual localization tasks; however, most of the work done concerns enabling localization in the first place. To be able to perform the internationalization of the product in as complete a manner as possible, it is a natural choice to contrast and compare existing solutions in internationalization.

The solutions picked for comparison function on slightly different levels. Whereas the *Zope Application Server* offers a complete solution for web application building, the *StringTemplate* template engine seems a little more abstract in its effort to truly externalize the handling of text output, and thereby also the localization and internationalization of the same. I will cast a look on the built-in internationalization support in PHP as well. Finally, as the product concerned in this work is built on PHP, I will perform a tentative peer analysis on an existing PHP product, The *Moodle eLearning* environment.

5.1 Zope Application Server

Zope is an application server written in Python, an object-oriented scripting language. The product is open source, and it is being worked on by a wide range of companies and developers throughout the world. The current version family of the application server is Zope 3, and at the time of this writing the most recent version is Zope 3.4. (Zope.org 2009.)

The internationalization options provided by Zope 3 mostly deal with making the textual portions of all code components and page templates translatable (von

Weitershausen 2007, 156). No attention at all is directed at collation or sorting, differentiated layouts for different locales, or text boundaries. The text tools are extensive, however. They enable very thorough localization of all visible textual elements in the application. The translatable text components in Zope are called messages which are treated as objects. Whenever a variable is needed as part of the translatable text component, a placeholder must be used in the component and the translation function must be called with mapping instructions as parameter (von Weitershausen 2007, 144-145).

Inherent in the text internationalization system is the concept of translation domains: it is a way to differentiate between different instances of identical text fragments that may appear in different contexts in the application. Usually the domain is a well-known name of a component or a page in which the message appears. It is used as a part of the ID of the message. (von Weitershausen 2007, 141-143.)

On a more practical level, Zope uses the *GNU gettext* system to store the messages in files called *message catalogs*. The freely available GNU gettext utilities are required to manipulate the message catalogs; the file format used by gettext is a binary one, so translated text files need to be compiled prior to use. Zope also comes with a special extractor utility which can be used to extract all translatable text portions from all application components. The utility generates text files which follow the format used by the gettext library. The application server uses a special directory structure inside a specific subdirectory in which the translated and compiled are stored. The storage locations are specified in the configuration options of the application. (von Weitershausen 2007, 157-158.)

The formatting of numerical values, date data values and so on is a task assigned to the localization functions of Zope. All values of these kinds of data are stored in an abstract format and displayed by means of a *locale* object. The number presentation does not offer a separate currency representation, however, so the decisions regarding the number of decimal places have to be implemented elsewhere. The date representation carries with it three length variations. The locale object is available through requests in different contexts, such as code fragments and page templates. (von Weitershausen 2007, 161-165.)

The most obvious drawback of the Zope internationalization implementation is that it is only available for the Zope Application Server and is not directly applicable in a PHP environment. There are some good ideas in the implementation though, such as the one behind the message ID's and the translation domains. The extraction utility is a very useful tool as well. Following this handling I will take a look at a Java-based internationalization implementation in a template engine called StringTemplate.

5.2 The StringTemplate Template Engine

StringTemplate is an open source template engine for the Java programming language. As a template engine, it can be used in generating formatted textual content such as web pages, e-mails or even source code. The engine has been ported to other programming languages as well: C# and Python. Its latest version number at the time of this writing is 3.2. StringTemplate lists support for internationalization and localization as its strengths. (StringTemplate Template Engine 2009.)

Parr's (2006, 65-67) treatment of page text localization in his StringTemplate template engine focuses much on so-called model-view separation that is strongly enforced by the engine. Although interesting and valuable as a concept when dealing with template engines, the separation is not in the main focus here. The handling of translatable text fragments as key-value pairs in property files is of more interest. Similarly to Zope's implementation, the property name (the "key") can include a page name, a sort of a translation domain fragment. Alternatively, each page can have its own separate property file. (Parr 2006, 66.)

Parr even demonstrates (2006, 66) a built-in piece of functionality in StringTemplate which enables designers and developers to manage the strings on the site itself via an administration interface. In this example, the strings are stored in a database and cached in memory while the application runs.

StringTemplate's treatment of the localization of data values such as dates and numbers relies on the concept of wrappers which are built in the code that deals with the logic rather than with embedding the formatting commands in the page templates. (Parr 2006, 67-69.) This is another example of the fact that Parr wants the separation between the

application's model and view layers to be enforced as strictly as possible. Although perhaps not directly applicable to the subject of this work, the concept of having easily callable methods in object instances referring to data values is an interesting one. The limited applicability follows from the fact that no templates are being used in the output creation process in the application platform, and a strict model-view separation does not seem as necessary in the given situation in the development team as Parr seems to think it is.

As in Zope, the most obvious weakness of this implementation is the lack of support for the PHP language. Another point to consider is that neither StringTemplate nor Zope pay any heed to locale-specific collation orders. StringTemplate's treatment of translatable text ultimately as key-value pairs is of interest, of course. The administration interface built into a web application is a fine example of usability of advanced features. In the next subchapter I will examine how PHP is equipped to handle the challenges of localization and internationalization.

5.3 Built-in and Extended Support in PHP

Dennis Popel (2007) claims that the internationalization support in PHP version 5 is a remnant from the older version 4 and as such is lacking in many things. He concludes his article in a promise that the next version, PHP 6, would correct many of these mistakes.

The application platform under construction is being built on the basis of PHP version 5, more specifically 5.2.6. In this subchapter I will examine the internationalization support available for the current version and not dwell very much on what has been promised of the future versions.

5.3.1 Built-in Features and Bundled Extensions

Popel (2007) identifies the `setlocale()` function as the main function in offering internationalization support. Having done this, he also lists problems concerning the use of the function. Firstly, the function is not platform independent as it relies heavily on the locale functions of the host operating system. This leads to the fact that to get the

function to work in the same, desired way between different server operating systems (that is, between Windows and Linux/Unix), special actions are needed. Secondly, and perhaps more importantly, Popel notes, as does the PHP documentation (PHP: setlocale - Manual 2009) on `setlocale()`, that the function does not work on a thread basis. This means, according to the PHP documentation, that if there is more than one person using the application at any one time and if the server software configuration is suitable for this, the function call might be initiated by one user and the other might see her application behavior changing as a result.

If the caveats listed for the use of the `setlocale()` function are acceptable or amendable, it is possible to use `localeconv()` to fetch default information concerning the selected locale. The PHP documentation for the function (PHP: localeconv - Manual 2009) lists several aspects concerning the locale that the function makes available to its user. These include thousands separators and decimal separators as well as properties concerning the output of monetary values. The `setlocale()` function's documentation lists further possibilities for locale use such as the `strcoll()` function for comparing strings, string manipulation functions such as `strtoupper()`, and time value output formatting with the `strftime()` function (PHP: setlocale - Manual 2009). Using the information provided by the `localeconv()` function it is also possible to format number output, to the extent of grouping the thousands and using the correct decimal separator, by way of the `number_format()` function.

Gettext is an extension of PHP that is included in the 5.2.6 version of the PHP package bundled in the version of Debian GNU/Linux, 5.0, used as the operating system platform in the new web application. The extension functions implement an API used for natural language support provided by the GNU Gettext (PHP: Introduction - Manual 2009). Despite GNU Gettext's many strong points, especially as regards to support for differences between natural languages, the extension's documentation links the use of the function `gettext()` to the use of the `setlocale()` function, so the same caveat regarding multiple users pertains to this extension as well (PHP: gettext - Manual 2009).

5.3.2 Extensions Provided by PEAR and PECL

PEAR is an acronym meaning “PHP Extension and Application Repository”. It is a repository providing open-source code libraries for the users of PHP as well as a distribution system for these libraries. Furthermore, PEAR includes PECL, the PHP Extension Community Library which contains binary extensions to PHP as opposed to the PEAR extensions which are written in PHP. (Manual :: What is PEAR? 2009.)

The PEAR repository offers two different internationalization extensions, called I18N and I18Nv2 (Package Browser :: Internationalization 2009). The latter of the two is the more recent one, its development having ceased at beta stage in 2006 (I18Nv2 2009). The less recent one is from the year 2003, also at beta stage and also not in active development any longer (I18N 2009). A quick peek into the extension documentation reveals that both extensions have been written for PHP 4. While not a critical disadvantage, since the original internationalization support does date to PHP 4 anyway, the changes brought on by PHP 5 in terms of object-oriented programming would require changes to existing coding standards in this project, an undesirable feature for an extension.

Moreover, neither of the packages includes any kind of input parsing for either dates or numeric or currency-related inputs. They do not take any stand on site-wide customizations in the manner StringTemplate does. Nor do the extensions offer any support for comparing strings, also known as collation. Both do offer something useful for building websites: an automatic way of negotiating the language and locale based on the HTTP headers sent by the browser. However, this is a feature not planned for the system. To achieve consistency, an implementation where the user stores her language and locale settings in the system is more desirable.

PECL has an extension to offer as well. The extension is known by the name Intl, short for International. The Intl extension will be bundled into the PHP distribution package in the upcoming version 5.3 (PHP 5.3.0RC1 Release Announcement 2009), but in the current 5.2.6 version the extension has to be manually installed using the PEAR installer.

The PECL/Intl extension offers a wide range of internationalization support. It functions as a wrapper for the International Components of Unicode (ICU) library, and its implementation resembles its counterparts in other programming languages such as Java and C++. (PHP: intl - Manual 2009.)

Intl supports collation, input parsing and message composition, that is, inserting correctly formatted values inside output messages, as well as functionality pertaining to acquiring information on different locales. The parsing functions all work as parts of the formatter classes. Having parsing methods means that it is possible to use the Intl extension to parse numerical inputs, date inputs and indeed any inputs for which a predefined format can be thought of. The formatter classes naturally also provide formatting for the output. (PHP: intl - Manual 2009.)

Although the Intl extension offers no direct support for locale-based site customizations either, its features regarding collation and, above all, input parsing, make it superior to any of the PEAR extensions in the case of this system. As regards the internationalization support required for the project management system, the PECL extension seems a very strong candidate for the foundation upon which the support could be built. Unlike the StringTemplate solution though, Intl offers no direct solution, at least not as an easily implementable built-in feature, for managing the strings. While this is undoubtedly a consequence of the differences of the levels on which these solutions operate, the PECL extension on a much lower level than the StringTemplate template engine, it leaves the problem of string management for the developer to solve.

5.4 Moodle eLearning Environment

Jevsikova (2006, 312) has analyzed three different open source virtual learning environments as regards their level of internationalization. One of the selected systems was Moodle, which is particularly interesting in this work since it has been implemented in PHP (About Moodle - MoodleDocs 2009). Jevsikova (2006, 313) treats the subject by selecting languages in three different character sets and by going through a list of issues specified in “international standard”[1]. Since the main object of our

[1]Original source: ISO/IEC 15897:1999 - Information technology -- Procedures for registration of cultural elements

interest is the success of Moodle in these tests, I will only treat the results as they regard it, and not the other two selected platforms. Also, I will not treat the issues that are not very specific to the environment of the project management system, such as the presentation of telephone numbers in the correct format, since there may be programmatical needs for locking down on a certain format to make it possible to connect the numbers to automated dialing, for example.

Out of a list of 16 elements, as seen below in table 1, Moodle only performs in an internationalized manner in 6 issues. Out of the six issues that Moodle does implement, the most interesting in this work, and not explicitly listed in the details of the previously treated internationalization implementations, is the color scheme and graphics tied to the locale.

Table 1: An implementation of internationalization issues (Jevsikova 2006, 316)

Internationalization issue	Moodle
Character encoding	+
National characters in login name	-
National characters in password	-
National characters in user's first and last names	+
Order of user's first name and last name	+
Matching of plural and singular forms	-
Grammatical name forms	-
Composition of several strings	-
Date format	-
Time format	+
First day of the week	-
Decimal separator	-
Thousands separator	-
Telephone number format	+
Dialog fields (controls) order	-
Color scheme and graphics	+

Jevsikova writes (2006, 313) that some of the issues listed in table 1, such as character encoding, decimal separator and thousands separator, are automatically dealt with by the server operating system. However, this is only the case in PHP if every request is run through separately and the locale setting function is not used again by another user

in the same server environment while the previous request is running, as pointed out in the previous subchapter. In spite of that, she does note (2006, 316) that the server software used, such as the PHP environment and the database system, may affect the results listed in the table.

No notion of configuring collation order by the alphabet used is presented in Jevsikova's work either. She points out (2006, 313), however, that allowing national characters in passwords and login names provide security and a more natural user experience. There is a point with which this can be countered, though: allowing national characters may hinder the use of the software in non-native environments as a user who has chosen national characters in her login details may have difficulties logging in from a computer not suited for inputting those national characters.

Jevsikova seems to take translatable strings for granted but does not offer any details as to how they are implemented in the different systems. She lists (2006, 314) parameterized strings as a point of interest though, as leaving them uncommented in the hands of localization vendors will lead to problems in the localization process. She stresses it is most important to comment the parameterized strings so that the word order can be arranged. There is another point, especially relevant to having numerical parameters inside such parameterized strings: Jevsikova points out that the commonly thought two noun forms matching numerical expressions, singular and plural, may not be enough in non-English languages. As examples she lists Russian and Polish which have three different noun forms.

Akin to matching numeric values and nouns, although not overly relevant in this work, are the grammatical name forms. Languages such as Lithuanian and Finnish are inflective languages in which the names of objects might have different inflected forms in different dialogs. A generator that could anticipate all possible names in all possible languages is outside the scope of this work, however. Jevsikova does suggest (2006, 314) that a placeholder for such a generator should be anticipated all the same.

Jevsikova fails to mention, as do the other implementers of the listed solutions except the PECL Intl extension, anything pertaining to the input of numerical or date-formatted values. As the project management system has been under heavy development for almost a full year now, controlling input has proven to be one of the key issues in

dealing with the information flow through the system. As we will see in the next chapter, a treatment will be offered, both in the server-side, PHP-based system and in the client-side JavaScript handling of input values.

6 Programming Practice and Translation Management

In this chapter I will apply the results obtained in chapters 4 and 5 with respect to what is said about the system setup in chapter 2. The results are divided into two main categories. Firstly, the tools that I selected for use in the system as regards the software platform; and secondly, the management of translation, that is the translatable content and its placement in the code.

6.1 Tools for Internationalization Chosen for the Platform

The first subchapter in the results is dedicated to the programmatic tools used in the software platform to provide internationalization. These are quite naturally divided into three subcategories, advancing from the uppermost tier of client-side programming in JavaScript and MooTools through PHP in the server side to the Database functionality performing the duties of data storage.

The order is not preserved in this subchapter, however, since some of the functionality provided by the server-side platform in PHP is necessary for the JavaScript implementation to work. The first tier I will handle is thus the server side programming in PHP. This will be followed by the handling of the JavaScript functionality and finally by a treatment of the necessities in the data storage tier.

6.1.1 The Server Side: PHP

The internationalization support has been implemented mainly in PHP. The client side of the platform uses the functionality and data provided by PHP. In this subchapter I will explain how the internationalization is implemented in the system and how it can be used in programming. The main application programming interface (API) is described first; then its uses are explained in more detail. Each of the explanatory sections is illustrated with code snippets of example usage. This section concludes with a look into how the client side code can gain access to the data and information provided by the API.

The I18n API

Most of the implementing code in the class providing the `I18n` API relies on the Intl extension provided by PECL and described in chapter 5.3.2. The API abstracts the internationalization implementation so that even if the underlying implementation would change, a thing not unheard of for such young products as the Intl extension is, the system would only require one change per each change in the implementation.

The `I18n` class offers various methods for internationalization. To get access to an instance of the class, the user uses the static `method I18n::getI18n()`, as shown below in code example 1. The class, in turn, uses the session data internally to determine the preferred locale for the session. The locale is used for all the methods of the class automatically, and there is no overriding mechanism in the method call signatures themselves.

In addition to `getI18n()`, there is another special static method `getMessageExt()` which skips the instantiation and just queries the message database for the requested string. The method takes the same parameters as the regular `getMessage()` method which is presented later in this subchapter.

```
$i18n = I18n::getI18n();
$extText = I18n::getMessageExt('ref.to.context', "Default text");
```

Code example 1: Usage of examples of two different static methods in the `I18n` class.

After the instance has been acquired, the user of the class can use all the other public methods provided by the class. These methods include:

- `formatNumber()`
- `formatDate()`
- `getMessage()`
- `sortAlpha()`
- `sortNumeric()`
- `parseNumber()`
- `parseDate()`
- `getJavaScriptLocaleInfo()`

What each of the listed methods is capable of is explained in the sections below.

Formatting Output

Numbers and dates, during storage and handling in the system, are stored as binary representations which strictly speaking have no forms of output at all as such. To format the output of such representations, the system uses the two corresponding methods in the `I18n` API: `formatNumber()` and `formatDate()`. While the latter only takes one parameter, the date to be formatted, the former accepts an additional one which signals the maximum number of fraction digits to be printed. Code example 2 below illustrates the use of these methods.

```
// Locale in session: fi_FI, which the instance is aware of
$i18n = I18n::getI18n();
echo $i18n->formatDate(time());      // Prints e.g. 12.5.2009
echo $i18n->formatNumber(1.132, 2);  // Prints 1,13
```

Code example 2: Usage of `I18n::formatDate()` and `I18n::formatNumber()`.

The `formatDate()` method accepts Unix timestamps (integers which carry information on how many seconds have passed since the Unix epoch) or results of the `localtime()` PHP function as parameters, just as the PECL extension class method it wraps, namely `IntlDateFormatter::format()`.

Apart from formatting numerical data, and perhaps offering the most visible and thereby most important aspect of output formatting, the `I18n` class also offers the user interface messages in the language set for the locale, provided that the language version exists in the message database. If the locale language is not available, the fallback is to use the English version; if even using that fails, the final fallback is to use the default text given by the caller of the `getMessage()` method.

The `getMessage()` method accepts three parameters, out of which two are required. The first two parameters, the required ones, represent the message identification string and the default message string, respectively. The third, optional, parameter is an array that holds the values used to replace the placeholders present in the original message text. Code example 3 demonstrates the use of the method.

```
$messageID = 'example.for.messageID';
// The curly braces denote the placeholder for a number
$default = "I see {0,number,integer} dogs.";
$replace = array(3);
// Chosen locale in this case: en_US
$i18n = I18n::getI18n();
echo $i18n->getMessage($messageID, $default, $replace);
// Prints: "I see 3 dogs."

$default = I see {0} dogs.";
$replace = array("many");
echo $i18n->getMessage($messageID, $default, $replace);
// Prints: "I see many dogs."
```

Code example 3: Usage of the `I18n::getMessage()` method.

Following the example of the method it wraps, the replacement placeholders are strings in curly brackets which hold at least the array index of the item which is to replace them. If the replacing item is not a string itself, the caller may specify instructions on how to handle the replacing value. In the example above, the number 3 is marked as an integer by concatenating the array index with the identifiers "`,number`" and further "`,integer`". The `getMessage()` method takes into account the locale specified in the class instance and returns a correctly formatted version of the text.

To satisfy demands for correct, locale-dependent comparison of strings, the I18n class offers to sorting methods `sortAlpha()` and `sortNumeric()`. The former of the two uses an alphabetic method in the sorting while the latter sorts numerically; the difference is perhaps best illustrated by the fact that `sortAlpha()` would in some cases sort two numerical values, for example 2 and 10, so that the new order would be 10, then 2, because the comparison would be done on a character-by-character basis. The other method would sort the numbers in the natural order: first 2, then 10.

These methods can be used on arrays of any objects that have a public method called `getFormDescription()`or a public property called `sortDescription`. Additionally, the methods support arrays of strings and numbers. They always return the sorted array and, depending on the optional second parameter, either rearrange the array keys or keep them tied to their corresponding values. By default, the array keys are rearranged, so the parameter must only be passed in case it is desirable for the keys to be preserved. Code example 4 shows different use scenarios for the methods.

```
// Locale: en_US
$i18n = I18n::getI18n();

$stringArray = array("Zebra", "Zulu", "Anchor");
// $result: array("Anchor", "Zebra", "Zulu")
$result = $i18n->sortAlpha($stringArray);

$test = array(3 => 2, 4 => 10, 5 => 11);

// $result: (4 => 10, 5 => 11, 3 => 2)
$result = $i18n->sortAlpha($test, true);

// $result: (0 => 10, 1 => 11, 2 => 2)
$result = $i18n->sortAlpha($test, false);

// $result: (3 => 2, 4 => 10, 5 => 11)
$result = $i18n->sortNumeric($test, true);
```

Code example 4: Different uses for `sortAlpha()` and `sortNumeric()` in `I18n`.

The sorting methods are useful in situations where a locale-specific sorting is desired, for example when ordering the options in a drop-down box, more commonly known in HTML as the `select` element.

At the moment, the system offers no tools for using locale-specific imagery or coloring. As the `I18n` API is a natural place for this, a future plan could be to expand the functionality to offer services for different types of images and coloring schemes.

Portable Document Format Output

The system uses an external open source library, *TCPDF*, for creating PDF output for different types of documents such as order confirmations and work orders. The library offers support for different font faces and bi-directional languages. This may be desirable in the future, when the language variety of the reports is extended.

A noteworthy matter is the fact that by default, the `I18n` and its methods only use the user's default locale to select the output language and conventions. In the PDF files this may not be acceptable as different partners may require communication in a language different from the one chosen by the user. Therefore, the previously mentioned public `getMessageExt()` method accepts a fourth parameter for setting the language. The parameter should be given as a string denoting the desired locale, for example `"fi_FI"`.

Validating and Parsing Input

As the output formatting section in this subchapter suggests, there are two types of information that need special attention since the internal representation is different from the presentation format shown to the user. These are numeric values and date values. Before the values given by the user can be incorporated in the data flow generated by the application, they have to be validated against their corresponding rules. This is because it would be impossible to perform any kind of logical operations based on data that is not of a syntactically correct internal representation for the operation.

Validating numeric and date inputs is a simple task provided by the `I18n` API. Because the input value has to be parsed as well, the method of choice is to incorporate the validation procedure into the parsing, as shown below in code example 5. So, calling either `parseNumber()` or `parseDate()` will return the Boolean value false when the validation fails.

```
// Locale: fi_FI
$i18n = I18n::getI18n();

$value = $_POST['the_number']; // Input value = "101,02"

if ( false === $parsed = $i18n->parseNumber($value) )
{
   // Error happened
}
else
{
   // Ok to go on with the value
   $newValue = $parsed * 1000; // The $newValue is 101020
}
```

Code example 5: The use of `I18n::parseNumber()`.

It is worth noting that the validation in `parseNumber()` can only accept two types of numeric values: the format which is the conventional one for the selected locale and the PHP default format. This is because setting multiple fallbacks could lead to “false positive” validations where the validation would pass but the parsed value would not correspond to the value entered by the user. The biggest problems arise from the arbitrariness of the separators used in the representations of numbers in different locales. Perhaps the most extreme examples are the conventional number formats of the `en_US` and `de_DE` locales, where the number representing “one million point one” would be written “1,000,000.1” and “1.000.000,1”, respectively.

The `parseDate()` implementation functions in a very similar manner but it only accepts dates in the format specified by the locale's conventions. This is usually not a problem, however, since the date inputs and outputs are tightly controlled in the system by the JavaScript-based date picker and output formatting.

Providing Data for the Client Side

The client-side code cannot cope with the internationalized features without some information from the server side. The JavaScript code needs information on the number and date formats to be able to parse the values correctly. If JavaScript were used for form validation, the information would be needed for that functionality as well.

The `I18n` API provides a method called `getJavaScriptLocaleInfo()` which essentially just returns an XHTML code snippet, a `script` element, which initializes the variables that the JavaScript version of the `I18n` class needs. The snippet also helps the date picker component, which will be described in more detail later, to choose the correct date format.

An addition to the services provided for the client side is the functionality in the Ajax server file in the application, `ajax.php`. When used with the correct `actionType` parameter, `getMessage`, the server file uses the `I18n` API's `getMessageExt()` method to retrieve the desired message in the correct language and returns the formatted message string to the caller, the functionality of which is also described in the next chapter dealing more closely with the client-side code.

6.1.2 The Client Side: JavaScript and MooTools

JavaScript code is code that is run in the user's web browser. In the project management system it serves, enhanced with the functionality offered by the MooTools platform, mainly as event listeners bound to clickable buttons and icons. There are some parts, though, where JavaScript functionality is needed to perform calculations on forms with a lot of numeric data or date data. In the system, input validation is handled server-side by PHP, so in this case I can omit internationalizing any validation procedures.

As may be apparent from the previous paragraph, the functionality in need of internationalization in JavaScript can be divided into two main groups: input validation and input parsing. Yet another issue in internationalization, however, is providing the messages in the user's preferred language. Since the validation is omitted in this treatment, I must concentrate on the other two aspects.

In practice the implementation is a JavaScript class which provides methods for acquiring translations and parsing inputs. As is the case in the PHP implementation as well, the class is called `I18n`. Its methods are called statically; no instantiation of the class is necessary for it to perform actions.

Translatable Content in Client-side Code

As in other programming, it is also wise in JavaScript to keep the translated content separate from the program code. The main problem with respect to the translation system as a whole is that JavaScript, as a client-side programming language, does not have direct access to the storage that holds the translated versions of the messages. This could be solved in a number of ways.

One way to solve the problem described above would be to create language-related message catalog files for each of the languages available in the system. This would, however, decentralize the management of translatable strings. Another, similar way would be to use the PHP interpreter to generate a language file based on the language setting present in the system. The file would have the same name for each language, only its contents would change. The problem with this approach is that, at least on basic settings, the user's browser will cache the message catalog file for future use and a language setting change might only take effect with a delay.

A third way to overcome the problems would be to use an asynchronous request to fetch the translated content from the server every time a message is needed. The server would know, based on the session it uses to store details on the user, which language is needed, and would be able to return the correct, and correctly formatted, string. The downside to this strategy is that it could potentially lead to a multitude of requests when there are a lot of messages to show.

A strategy similar to the Ajax-based one would be to include the messages needed on the current page in a script element on the page that is being shown at the moment. This would however require that the server-side part would somehow know which messages are needed on which page, which would not lend well to the manageability of the code. Another alternative would be to simply include all messages that the JavaScript code could need into each page. This would increase the size of the page somewhat but the texts would be available to any script that would need them. A downside in this strategy is that a method of formatting the messages, for example by injecting variables in place of placeholders, would be needed while the same would be readily available in the Ajax method on the server anyway.

As a conclusion for the treatment of translations in JavaScript, the method dubbed ‘The Ajax Method’ is the one to choose in this project. To avoid duplicate implementations, and since the server-side of the platform already has similar functionality available, I choose not to implement a full-fledged message formatter in JavaScript. Whenever a piece of JavaScript needs a message, it can therefore statically call the `I18n` class method `getMessage()` as shown in code example 6 below.

```
var helloMsg = I18n.getMessage('contextInformation', 'Hello!');
alert(helloMsg);
```

Code example 6: Creating and alerting a localized message using the I18n class.

As code example 6 illustrates, the `getMessage()` method takes on at least two parameters: the information on the context, serving as a message ID in the JavaScript environment, and the default string which is output in case the request for the translated version fails. In case the request fails for a technical reason though, the default message receives no handling at all. This means that any placeholders will remain as they are in the default text. The limitation is due to the fact that no formatting engine is implemented in the JavaScript `I18n` class itself.

As in the PHP version of the corresponding method, there is an optional third parameter though, and it deals with inserting any calculated values into the message. The third parameter, when given, must be an array the values that are meant to replace the placeholders in the resulting text. Code example 7 below illustrates the use which is very similar to that of the PHP version of the same method.

```
var testArray = [1, 2, 3];
var replacements = [testArray.length, "Kalle"]
};
alert(
   I18n.getMessage(
      'testContext',
      "Hi {0}! Your array has {1,number,integer} members.",
      replacements));
```

Code example 7: Usage of the optional third parameter of `I18n.getMessage()`.

The code example also shows that it is not necessary for the replacements to hold literal strings for the method to work. Any value that can be represented as a string, even a function that returns a string, can be used.

Having dealt with translatable content in the client-side script, I will now focus on how to accept different forms of input while still being able to calculate values based on the inputs.

Date Inputs

To assist in keeping the input of dates simpler and to keep the format correct, the application uses a third-party date picker. In a normal case the user has no possibility to manually change the value of a date input; the value is always controlled by the date picker which in turn receives information on the correct date format from server-side.

Parsing Input Values to Enable Calculation

The system uses JavaScript as a tool on some forms to calculate dates or sums based on data provided by the user. Since the output on the forms is formatted as detailed above and since the system should support input in the default format provided by the locale, parsing the inputs presents a problem very similar to that in PHP.

As in PHP, in JavaScript a native floating point number is composed of the integer part, without any thousand separators, separated from the fraction part by a dot. JavaScript sees the values in text input boxes as strings but they can be converted into floating point numbers as long as they fulfill the requirements set for strings representing numbers. For this, the native `toFloat()` method that the String objects provide can be

used. If the value contains other characters than those specified as parts of a number, the method returns a special value `NaN` (Not a Number).

The solution to the parsing problem seems natural. The `I18n` class provides a static method called `parseNumber()` which takes a string as a parameter and tries to parse it into a floating point number. The method uses information on the user's preferred locale provided by the server to help in parsing in a correct manner. The method returns the parsed string which can be then cast to a number using the native `toFloat()` method. The result of the cast will still have to be checked with the native `isNaN()` function but since the string has been parsed, the local variant of a floating point number is correctly recognized as a number anyway. Code example 8 below will demonstrate the use.

```
var intlFloat = '1.000,3'; // German number format
var parsedFloatString = I18n.parseNumber(intlFloat);
var parsedFloat = parsedFloatString.toFloat() // 1000.3
var result = parsedFloat+1.1; // 1001.4
alert(I18n.formatNumber(result, 2)); // alerts 1001,40
```

Code example 8: Basic usage of the `I18n.parseNumber()` method.

The code example above also demonstrates the use of another `I18n` method, the `formatNumber()` method. The method takes one compulsory and one optional parameter. The first one, the compulsory one, is the number to format according to the locale rules. The second one is an optional one that defaults to 0. It signals the number of fraction digits to use in the formatted number. The formatter only replaces the decimal separator as the MooTools platform lacks the ability to format numbers in as nice a manner as it can format date output.

The formatting for numbers could be fetched asynchronically from PHP on the server side but this would require all too many requests as the forms handled tend to be large and as each number would have to be formatted as is. A better compromise as regards performance is to drop the thousand separators completely when formatting output.

Date formatting and value parsing face a similar issue to that of numbers, although the default format is not as user-friendly or simple. However, the methods provided by the `I18n` class provide similar functionality and are named similarly: they are called `parseDate()` and `formatDate()`, both in the `I18n` namespace.

Internally, the `I18n` class parses and formats the data based on the information it gets from the PHP tier. As shown earlier, the server-side tier inserts a short script element in the header of each output file that contains the information needed for parsing and formatting date and number data. Based on this data, the `I18n` methods use the MooTools extensions of the `Date` class for date data and regular expression matching for numeric data to perform the parsing. The formatting is provided by the `Date` class extensions in MooTools and the native `String.replace()` method, respectively.

6.1.3 Database

The Database tier is, internationalization-wise, a simple one to handle in the system since the collation services are provided by the `I18n` API in the PHP tier. This means that all that is needed as regards the database is to keep the communication between the storage and the program code in UTF-8 character set and to store the data in UTF-8 in the database.

The communication can be set to default to UTF-8 in one of two ways: either each connection made by the PHP tier to the database tier must instruct the server to respond in UTF-8 or the default can be set in a configuration file used for the server. In this case, the database handling API in the PHP tier handles the character set setting per connection. The `I18n` API is not needed in this, unless at some time it becomes necessary to change the character set. If that is the case, it is a simple change in the database API to change the character set setting to conform to a value given by the `I18n` API.

6.2 Managing Translatable Content

The concept of message catalogs explored in handling the Zope Application Server is present in the project management system as well. The catalog is the storage for the translated content for the system. In this subchapter I will concentrate on the incorporated tools to keep the message catalog up to date.

The message catalog is located in a simple database table listing the ID, the language version, the additional contextual information and the content of the message. This

method of storage was chosen over the GNU gettext implementation for the simple reason that in the context of the development, there was not enough time to learn the use of gettext. An additional factor was the fact that in storing the messages in the simple database, it is a rather simple task to develop tools to manage the strings by way of PHP functionality and to manipulate them so that localizing the textual portions is as effective as possible. Also the caveats concerning the vulnerability for concurrent use, as listed in context with the built-in internationalization features, are worth keeping in mind.

6.2.1 Keeping Translatable Content Up to Date

Applications have the tendency of developing over time, and this brings challenges to the management of translatable content. Translating texts costs money, and having as little as possible to translate is the optimal way to keep the costs in check. Additionally, some textual components might fall out of date and their default texts might be revised when developing the application further. These changes should be reflected in the translated content as well, so the source material should be kept up to date.

For now, the system used to update and create new translatable content has relied on manipulating text files containing source code in the SQL language, meant for manual insertions of new texts and erasing the old ones. This is a method that has grown the more cumbersome the more time has passed since there is no way of knowing which texts are still in use and which are not.

To solve the problem, a system is needed that would at least keep track of which strings are in active use in the application. In addition, the system would have to enter newly created strings into the message catalog. In this respect, the concept of the message collector in the Zope Application Server would be optimally usable. However, since in the case of this application that is not directly usable, an alternative has to be considered.

Since time will not permit developing a text parser that would analyze all possible uses of the strings present in the source code, another kind of a method is needed. In this case the solution is a debug mode for the application. By using the application in the debug

mode, the developer is able to wander through different pages, optimally using a site map. The application will gather all the messages encountered in the debug mode into an intermediate message catalog, a database table where it adds information on the date of last access, the default string used in invoking the message, and the current form of the text in the source language, English, that is present in the message catalog.

Based on the data in this intermediate table, a rather simple interface could be built that could be used to update the translatable content. At a later stage, this interface could also be used in exporting and importing translation files from and into the message catalog. However, since the development of the internationalization API has taken so long, the interface, and the whole system for updating the strings, is still under development.

6.2.2 Providing Context for Strings

For purposes of more reliable and higher quality translation, at least some kind of information on the context in which the translated text is to be used is very useful. Providing this information is possible in the current message catalog system as well, since there is a place for this information in the SQL template for each string. On the other hand, even the message ID given in the database provides some useful information on the context since the ID is a textual one and composed of information on where to find the string in the first place. For example, a string denoting a save button on a form concerning a project's information could have the ID 'Project.views.form.buttons.OK'.

In the system proposed for updating the texts, the interface would also include a possibility to write down the context information before saving the translatable content.

7 Conclusions

The main goal for the thesis was the exploration of internationalization and localization. It should be added, though, that the exploration should be done with respect to the system being developed. This exploration has been carried out in an extensive, yet not exhaustive, manner in chapter 4. The supporting aims of explaining the environment have been met in chapters 2 and 3, while the benchmarking was carried out in chapter 5.

The benchmarking section left something to be hoped for as none of the printed sources even so little as mentioned anything about accepting user input in her native language. Fortunately, the PECL Intl extension made it possible for me to support locale-dependent input in the system, at least to some extent.

The results in chapter 6 were rather successful when it comes to the application programming interface both in the server-side code and the JavaScript code. The practical code examples help the developers to implement internationalized functionality in a coherent manner. However, the system planned for keeping the content up to date could not be implemented in time, so only a plan for it was presented in the sixth chapter.

As regards the future development of the application, the API is still left without support for culture-dependent coloring and imagery. In addition to those, the extensive support for linguistics offered by the GNU gettext library should at least be studied further. Now the time limits would not permit a study period long enough to make full use of the system; also the compatibility with different computer aided translation tools should be taken into account. It could be easier to support the most commonly used tools with a custom implementation where the texts are stored in a database catalog rather than in the file structure the gettext suite would require. In case the gettext suite is deemed unusable, the system for managing and updating the translatable content is a much desired addition as a development tool.

Parsing numerical data in a correct manner proved to be a problem since the formats used throughout the world differ so wildly from each other. In the future, one of two paths should be chosen as regards parsing and validation: either standardize the input format, and the number formatting used to populate the input when using previously

saved values; or develop further the recognition, validation and parsing logic so that it can be used with more fallback options than just one as it is now. Whichever route is eventually chosen, it will also enable the use of client-side validation of inputs in the forms. This kind of approach for input validation would undoubtedly improve the user experience as the application would feel more responsive and intelligent to the user.

Although the results acquired only apply directly to the specific web application platform presented as the basis of this work, the analysis and benchmarking can be used as references as to what the rather vague term internationalization actually conveys. The brief glance cast at the process of software localization can also be an adequate starting point for developers seeking knowledge on localization as well as on internationalization.

The most important finding in this work as regards its subject, internationalization in PHP web applications, is the discovery of the Intl extension, becoming available as a bundled extension in the upcoming version 5.3. As the extension and its documentation mature, it will become a good starting point for internationalization ventures in web applications utilizing the current version of PHP. This way the developers will not have to wait for PHP 6, the development of which seems to be dragging out still.

References

Printed Media

Appu, Ashok 2002. Administering and Securing the Apache Server. Cincinnati, Ohio, USA: Premier Press.

Converse, Tim; Park, Joyce; Morgan, Clark 2004. PHP5 and MySQL® Bible. Indianapolis, IN, USA: Wiley Publishing, Inc.

Esselink, Bert 2000. A Practical Guide to Localization. Philadelphia, PA, USA: John Benjamins Publishing Company.

Hogan, James; Ho-Stuart, Chris; Pham, Binh 2004. Key challenges in software internationalization. ACM International Conference Proceeding Series; Vol. 54 Proceedings of the second workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation - Volume 32. Darlinghurst, Australia: Australian Computer Society, Inc.

Jevsikova, Tatjana 2006. Localization and Internationalization of Web-Based Learning Environment. In work Roland Mittermeier (ed.) Informatics Education - The Bridge between Using and Understanding Computers. Berlin-Heidelberg, Germany: Springer-Verlag.

Newton, Aaron 2008. MooTools Essentials: The Official MooTools Reference for JavaScript™ and Ajax Development. Heidelberg, Germany: Springer-Verlag GmbH & Co. KG.

Parr, Terence 2006. Web application internationalization and localization in action. ACM International Conference Proceeding Series; Vol. 26 Proceedings of the 6th international conference on Web engineering. New York, NY, USA: ACM.

Sheldon, Robert; Moes, Geoff 2005. Beginning MySQL®. Indianapolis, IN, USA: Wiley Publishing, Inc.

Snook, Jonathan; Gustafson, Aaron; Langridge, Stuart; Webb, Dan 2007. Accelerated DOM Scripting with Ajax, APIs, and Libraries. New York, NY, USA: Springer-Verlag New York, Inc.

von Weitershausen, Phillip 2007. Web Component Development with Zope 3. 2nd edition. Berlin-Heidelberg, Germany: Springer-Verlag.

Yang, Yanxia 2007. Extending the User Experience to Localized Products. In work Nuray Aykin (ed.) Usability and Internationalization. Global and Local User Interfaces. Berlin-Heidelberg, Germany: Springer-Verlag, 285 - 292.

Electronic Media

About Moodle - MoodleDocs. [www page]. [referred to 22.4.2009] Available: http://docs.moodle.org/en/index.php?title=About_Moodle&oldid=51711

Effects of Locale - The GNU C Library 2009. [www page]. [referred to 6.5.2009] Available: http://www.gnu.org/software/libc/manual/html_node/Effects-of-Locale.html

GNU `gettext' utilities. [www page]. [referred to 6.5.2009] Available: http://www.gnu.org/software/gettext/manual/gettext.html

I18N. [www page]. [referred to 4.5.2009] Available: http://pear.php.net/package/I18N

I18Nv2. [www page]. [referred to 4.5.2009] Available: http://pear.php.net/package/I18Nv2

Manual :: What is PEAR?. [www page]. [referred to 4.5.2009] Available: http://pear.php.net/manual/en/about.pear.php

MySQL 5.1 Reference Manual :: 12.7 SQL Syntax for Prepared Statements. [www page]. [referred to 29.4.2009] Available: http://dev.mysql.com/doc/refman/5.1/en/sql-syntax-prepared-statements.html

MySQL 5.1 Reference Manual :: 19 Stored Programs and Views. [www page]. [referred to 26.4.2009] Available: http://dev.mysql.com/doc/refman/5.1/en/stored-programs-views.html

Newton, Aaron 2009. More To Love. [online] [referred to 25.4.2009]. http://mootools.net/blog/2009/03/09/more-to-love/

PHP 5.3.0RC1 Release Announcement. [www page]. [referred to 4.5.2009] Available: http://www.php.net/archive/2009.php#id2009-03-24-1

PHP: gettext - Manual. [www page]. [referred to 1.5.2009] Available: http://www.php.net/manual/en/function.gettext.php

PHP: intl - Manual [www page]. [referred to 4.5.2009] Available: http://www.php.net/intl

PHP: General Information - Manual. [www page]. [referred to 27.4.2009] Available: http://www.php.net/manual/en/faq.general.php

PHP: Introduction - Manual. [www page]. [referred to 1.5.2009] Available: http://www.php.net/manual/en/intro.gettext.php

PHP: localeconv - Manual. [www page]. [referred to 1.5.2009] Available: http://www.php.net/manual/en/function.localeconv.php

PHP: setlocale - Manual. [www page]. [referred to 1.5.2009] Available: http://www.php.net/setlocale

Popel, Dennis 2007. i18n with PHP5: Pitfalls. [online] [referred to 1.5.2009]. http://www.onphp5.com/article/22

Proietti, Valerio 2009. MooTools - a compact javascript framework. [www page]. [referred to 25.4.2009] Available: http://www.mootools.net

Proietti, Valerio 2009. MooTools 1.2.2 and the New MooTools More. [online] [referred to 25.4.2009]. http://mootools.net/blog/2009/04/23/mootools-122-and-the-new-mootools-more/

StringTemplate Template Engine. [www page]. [referred to 20.4.2009] Available: http://www.stringtemplate.org/

The XMLHttpRequest Object. [www page]. [referred to 25.4.2009] Available http://www.w3.org/TR/XMLHttpRequest/

Zope.org. [www page]. [referred to 18.4.2009] Available: http://www.zope.org/