Authors' Information

Jose A. Calvo-Manzano – e-mail: jacalvo@fi.upm.es
Gonzalo Cuevas – e-mail: gcuevas@fi.upm.es
Ivan Garcia – e-mail: igarcia@zipi.fi.upm.es
Tomas San Feliu – e-mail: tsanfe@fi.upm.es
Ariel Serrano – e-mail: aserrano@zipi.fi.upm.es

Universidad Politecnica de Madrid – Facultad de Informatica, Campus de Montegancedo s/n, 28660 Boadilla del Monte, Madrid, Spain.

Magdalena Arcilla - Universidad Nacional de Educacion a Distancia – Escuela Tecnica Superior de Ingenieria Informatica, C/ Juan del Rosal 16, 28040 Madrid, Spain; e-mail: marcilla@issi.uned.es

Fernando Arboledas – e-mail: fernando.arboledas@madrid.org
Fernando Ruiz de Ojeda – e-mail: frdo2@madrid.org

Informatica y Comunicaciones de la Comunidad de Madrid (ICM), C/ Embajadores 181, 28045 Madrid, Spain.

A PRACTICAL CASE OF SOFTWARE LOCALIZATION AFTER SYSTEM DEVELOPMENT[1]

Jesus Cardenosa, Carolina Gallardo, Alvaro Martin

***Abstract**: Internationalization of software as a previous step for localization is usually taken into account during early phases of the life-cycle of software development. However, the need to adapt software applications into different languages and cultural settings can appear once the application is finished and even in the market. In these cases, software localization implies a high cost of time and resources. This paper shows a real case of a existent software application, designed and developed without taking into account future necessities of localization, whose architecture and source code were modified to include the possibility of straightforward adaptation into new languages. The use of standard languages and advanced programming languages has permitted the authors to adapt the software in a simple and straightforward mode.*

***Keywords**: Localization, Internationalization, XML.*

***ACM Classification Keywords**: D. Software, D.2.7 Distribution, Maintenance and Enhancement*

Introduction

Any technical device devoid of human interaction operates and yields an expected level of productivity regardless of the cultural environment where it is located. The same can be said for software, as long as it does not call for any human interaction. However, many software applications require human interaction for a correct functioning. In this case, the level of productivity of the software will depend not only on software's intrinsic technical characteristics but on external human factors.

When a software application is used in a context with a different cultural environment (like different mother language, different icons, symbols, etc.) from its original one, a process of adaptation into the new work culture is required. This process is known as **localization**. The adaptation into a new culture not only comprises evident factors like the language of the interface and messages to the user, measure units or data formats (also known as overt factors according to [Mahemoff et al, 1998]); but also other slippery and fuzzy issues that finally

[1] This paper has been sponsored by Universidad Politécnica de Madrid and Autonomous Community of Madrid through the project EXCOM (R05/11070).

distinguish a culture, like mental disposition, perception of the world, rules of social interaction, religion, etc., which are referred to as the covert factors of a culture. More specifically, the process of localization consists on the "**adaptation** of a product, application or document content to meet the language, cultural and other requirements of a specific target market (a *locale*)", as expressed by the W3C [W3C, 2005].

On the other hand, **internationalization** refers to the design and development of a product, application or document content that **enables** easy localization for target audiences that vary in work culture, region, or language. In this sense, it can be said that internationalization precedes and facilitates the task of localization.

Besides, the processes involved in localization of software applications changes significantly depending on whether it is done over a pre-existent application or over a developed application.

The next section sketches the most frequent practices of software internationalization and localization in software design. However, in pre-existent applications, and depending on the system development methodology, the localization process can become very expensive in terms of time and resources. We will show how we internationalized and subsequently localized a pre-existent application in a cheap and quick manner, by means of advanced standard implementations languages like Visual .Net and XML.

Software Architectures for Internationalization and Localization

As we commented in the previous section, the internationalization and localization (I&L) processes deal with more that mere language issues. However and for the purposes of this paper, we will consider only the language adaptation, which is the most prominent and visible aspect of I&L.

Apparently, an **internationalized** product does not entail structural changes in order to adopt a new language. Internationalization consists on abstracting the functionality of a product of any given language, in a way that the support of the information of the new language can be added afterwards, without facing the source code (dependent of a given language) when the product is localized into a new language. Currently, main development platforms offer support and tools to facilitate the internationalization of over factors of applications [Hogan, 2004], [Huang et al, 2001], in a way that currently problematic questions are centered on the optimization of the internationalization processes within the life cycle of the application.

There are <u>three</u> main approaches for internationalizing an application. The <u>first</u> one is the system where messages, menus and other culture-sensitive factors are embedded in the source code of the application. This approach obliges to develop a different version of the system for each of the target cultural environments. Each version requires independent process of testing, maintenance and upgrading, multiplying the costs of localization.

The <u>second</u> approach consists on extracting messages to the user of a given application into an external library. The application is generated from a common source code that links to the culture-sensitive libraries. Although this architecture resorts on a unique source code, only the languages contained in the external library could be incorporated, and it is required to test and maintained each of the supported languages individually.

The <u>third</u> and last approach consists on an architecture composed of the core of the application comprising all the functionalities but independent of cultural factors, which dynamically access to files of external resources that contain information about the corresponding culture (localization packages). The difference with the previous approach lies in the fact that the culture-independent code dynamically calls to the information of culture, so that only one executable must be tested and maintained. Once the set of supported cultures is tested, the addition of new cultures does not imply modifications. From this general idea, each author develops his/her own way of acting. For example, [Stearns, 2002] describes the process of developing systems sensitive to cultures using JAVA and XML for resources files, whereas the environment GNU/Linux [Tykhomyrov, 2002] and the Free Software Foundation [FSF, 2002] prefer the use of special libraries that facilitates the extraction of the localizable contents of the application and the construction of localization packages.

Regarding the aspects related to the life cycle of the internationalized software, [Mahemoff et al, 1999] presents a methodology for requirements specification to develop culture-sensitive systems. On the other hand, [Huang, 2001] offers a description of the processes to be followed to create culture-sensitive software, emphasizing the fact that the internationalization tasks should be included in the corresponding phases of the life cycle of software.

The work on the area of localization is complemented with research on the problem of localizing software already internationalized. Even when the technical procedure for software internationalization is optimized, the bottleneck lies in the **localization** processes of a product. The process of internationalized software localization resorts on the concept of repository and reuse of translation resources. That is, apart from the external file that contains the messages to the user and its translations, there is a repository where translations are stored for their subsequent reuse. In some cases, there are also repositories for terminology.

The following standards have been established to facilitate the task of managing the culture-sensitive resources files and their communication with repositories:

- XLIFF (XML Localization Interchange File Format) defines a standard format for resources files that stores the translated strings, in a way that tools for assisted machine translation can be developed independent of the application to be localized, as well as transporting the translation information from one phase of the process to the following phase [OASIS, 2003].
- TMX (Translation Memory Exchange), allows for the storage and interchange of translation memories obtained after the use of automatic tools for translation [LISA, 2005].
- TBX (Term Base Exchange), defines a standardized model for terminological databases [LISA, 2003].

There are also some common practices among companies that have become a "*de facto*" standard [Hogan, 2004] aiming at minimizing the impact of localization on commercial software products, namely:

- Extraction of the fragment of texts used in the user interfaces into resources files.
- Control of the extracted texts, contexts and their translations.
- Outsourcing of the translation tasks to specialized companies.
- Simplification of the contents of the chains and their contents as a previous step to the sending to translation centres.

However, as can be seen, internationalization architectures and localization standards do not offer a solution for already existent applications that require international dissemination. That is, according to these architectures, localization is a bottleneck and it is only possible with an internationalized architecture. But what happens if we want to adapt a software application into many languages? The next section presents how an architecture can be changed in an afterwards-mode and how we internationalized and subsequently localized a pre-existent application in a cheap and quick manner, by means of advanced standard implementations languages like Visual.Net and XML.

Internationalizing an Existent Application: the Context

The starting point of this work is a software application for multilingual generation that allows for human interaction. It is an interactive application composed of a user interface where the user can manipulate semantic representations of the text to be translated.

The only requirement in the development of this tool was the use of UNICODE files, because of the almost certainty that the tool was going to be used for analysis and generation in a variety of languages. This obviously involved the future necessity of localization of the tool. It seems clear that the internationalization should be foreseen and reflected at the level of requirements specification and that it consists on something more than the mere use of UNICODE files. We will show how, in cases where internationalization has not been taken into account in the development processes, a pre-existent system can be adapted a posteriori for internationalization purposes. That is, our work is framed in the following context:

- There is a need of a future internationalization and localization processes, which is partly reflected on the requisites through the need to work with UNICODE files.
- The system is implemented in a development framework compatible with the use of UNICODE files (VB.NET), which guarantees the strict observation of the previous requisite, but nothing else.
- Apart from UNICODE files, there is not any other feature in the system oriented towards internationalization and subsequent localization.

The result of this situation is an environment able to import and deals with UNICODE files, that works with several languages (it is a translation aid tool) but with the totality of the user interfaces functionalities in just one language (in this case, Spanish).

The internationalization process that we are going to describe has been carried out **after** the complete development of the tool, proving that at least on of the most important and basic tasks of localization, such as language adaptation, can be done even without having internationalized the system in previous development phases.

Description and Preliminary Analysis of the Pre-existing System

The application is conceived as an environment for linguistic tasks, in which some external resources and components as language analyzers, language generators and dictionaries are integrated, with a powerful user interface and graphics management. From the architectural point of view, there are three main subsystems in the environment, which are:

- Kernel: it is the component in charge of managing most of the information and data flow of the application, as well as integration with external language analyzers, generators and dictionaries.
- Graphic controller: this component is in charge of managing the graphical display of abstract and semantic structures, as well as the correspondence between semantic structures and graphics.
- Interface: this component manages the communication of the application with the user. It mainly consists on the user interface with a few functionalities, which are delegated to the kernel or the graphic controller.

Figure 1 shows the application architecture and information flow graphically.

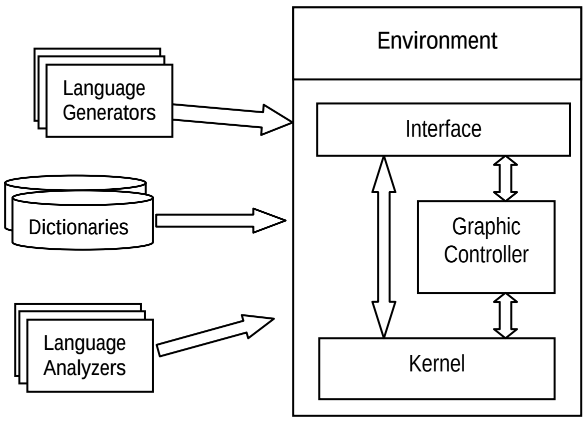

Figure 1. Architecture of the application

The entire interface is in Spanish. It is important to note that there are two types of textual elements in the application interface: “message errors” occurring in unexpected situations (also called emerging messages) and the text of the environment itself.

Each subsystem can generate a given number of emerging messages and windows with their corresponding text elements. In this way, the textual elements are scattered all over the source code.

A preliminary analysis of the source code shows that the textual elements follow two regular patterns. The first pattern corresponds to “emerging messages”. These are created with the statement “*msgbox ()*” (an abbreviation for *message box*); the text assigned to the emerging message is written within the parenthesis. As an illustration, a real emerging message informing of a file that is not found will have the following code:

```
msgbox("Fichero no encontrado")
```

The second pattern corresponds to the text used in windows and buttons of the application, which have the general pattern:

```
component.text = "Text associated to this component"
```

Where the expression “component.text” is the convention in VB.NET to note that the string in double quotes is the text that will appear on that specific component. For example, to assign the text “Aceptar” (OK) to a given button, we write:

```
button.text ="Aceptar"
```

Since we are going to restrict the I&L process to just linguistic issues, these textual elements will be the subject of the I&L processes.

Strategy for I&L, Conceptual and Architectural Design

Our specific problem is the need to adapt the environment into the English language. The most obvious and even quick solution is to search for all the text elements in Spanish and create a new version of the application with the interface in English. However, there are some requirements on the I&L adaptation, such as:

a) The localization process should be done by translators / final users.
b) Maintainability of the system and translations should be guaranteed.
c) It is desirable to produce a core application abstracted from the linguistic issues.
d) The pre-existing components must not be functionally modified.

Therefore, the architecture should be modified with the addition of a new component in charge of the internationalization functionalities; so that the textual contents of new languages are stored as a new resource (in the form of an external file, for example) which can be read and processed by the application itself.

The new component is in charge of reading the external files with the translations of the textual elements and imports them into the environment so that messages and interfaces can be shown in different languages. The result is a new software architecture as illustrated in figure 2.

Thus, the global strategy promotes the creation of a new specific component that once integrated in the original architecture is responsible for all the internationalization tasks. The basic functionalities of this new component should be:

1. Identification and labelling of all the text strings written in Spanish language of any kind (emerging messages, buttons, windows, and any other textual elements)
2. Extraction of these strings and generation of a XML file according to a predefined structure
3. Capture of the new XML file, once all the identified strings have been translated into the new language in the XML file.
4. Insertion of the translated strings according to the labelling.

The detailed description of this process is shown in the next section.

The Practical Case

The new component, called "Internationalization Manager", serves a number of functions that guarantee that the required language changes are carried out over the existent environment, while intervening in the current software as less as possible. Figure 2 shows the new architecture of the environment and how the "Internationalization Manager", together with its functional element the Text Management Module (TMM), is integrated in this new architecture. In the remaining, we will describe how the new component works and its main functionalities.

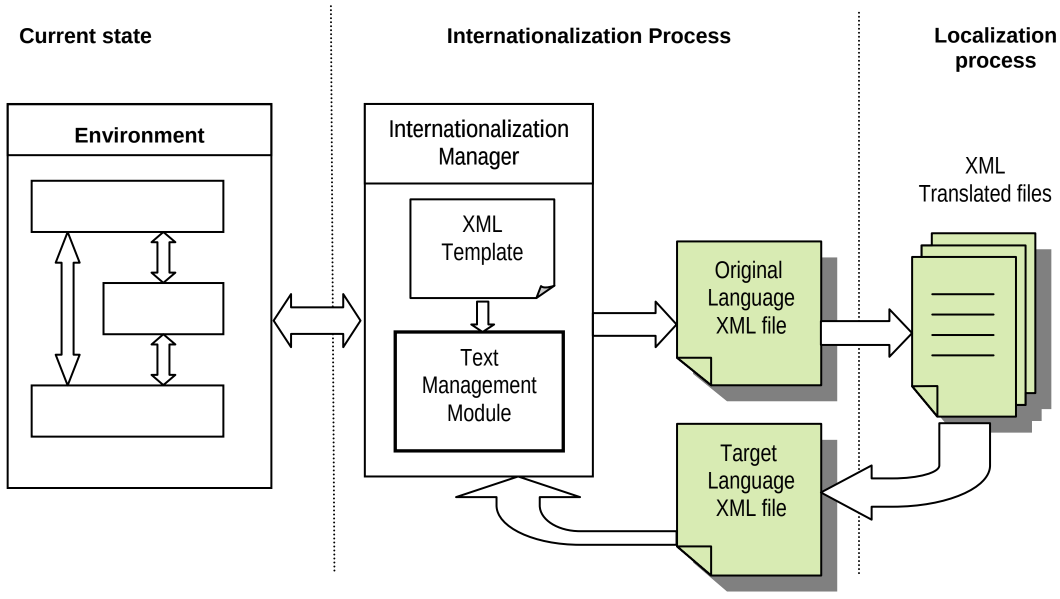

Figure 2. Global process

1. Text string identification and labelling.

This first functionality consists on identifying the textual elements in the original language (in our case, Spanish) following the two aforementioned search patterns, namely `msgbox` and the `component.text`. This function has been carried out by means of a script that identifies these text strings. The result of the script is a file where not only the text string is stored, but also additional information associated to the string, like its location, the component it belongs to, and other information that could be useful. All the information that the script gathers about a text string is labelled with a numeric identifier. The only modification that is done from this moment over the original software is the substitution of these strings by a function that calls for the identifier in the XML file of the required language and inserts the text string contained in the XML file. Let's see an example of how it works.

Suppose the source code of the application in Spanish contains the following an emerging message:

`msgbox (`*`“Error: Archivo no encontrado”`*`)` (English: “Error: *File not found*”)

The Spanish text string is substituted by the following:

`msgbox (`**`InternationalizationManager.GetText(57)`**`)`

Where `InternationalizationManager` is the function that calls to the corresponding component of the TMM that executes the instruction `GetText(57)`. This instruction captures and temporally inserts the text string labelled with the identifier `57` in the language selected by the user in its place. Currently there are not text strings of a specific language in the environment anymore but functions like the aforementioned that allow for the incorporation of a new language in the environment without further changes over the original software.

2. XML structure

The information about the text should be structured according to an XML template that permits to save a unique structure but modifiable in the data (in our case the text translations) that guarantees their interchangeability and maintainability. This XML file can be imported by the environment since the programming language (VB.net) is provided with an XML parser. This XML file is delivered to the translators and looks like as shown in figure 3.

The first line of the XML file indicates the version of the XML standard being used and the type of codification of the file (UNICODE in this case). The second line has an empty attribute `langID=""` that will indicate the target language of the translation of the strings. The rest of the XML file is divided in three elements `<userInterface>`, `<kernel>` and `<graphicController>`, each pertaining to the main components of the software. Each component is composed by a number of `<item>`. An `<item>` stores the following elements:

- The attribute `“id”` (in the example of figure 3, one “id” is 56) that uniquely identifies the linguistic text element and its presence in the software component.
- The `“orig”` attribute corresponds to the text string in the original language. One example is the Spanish string *“¿Desea continuar?”*.
- The `<translation>` element which is empty and will have to be filled with the translations into the target language.

This file is distributed to translators so that they can perform the translations tasks in their corresponding working places, allowing for an absolute independence of the translation process and its integration in the software environment. The XML files in the target languages are delivered to the TMM and located in the corresponding directory so that they can serve as the different language options of the environment to be selected by the user. An example of an XML file containing the translations for English is shown in figure 4. This file is the result of the localization process.

```
<?xml version="1.0" encoding="UTF-8"?>
 <localisation langID="">
  <userInterface>
    <item id="56" orig="¿Desea continuar?">
       <translation> </translation>
    </item>
       <item id="57" orig="Error: Archivo no encontrado">
       <translation> </translation>
    </item>   ...
  </userInterface>
  <kernel>
    </item>
       <item id="64" orig="Atributo no válido">
       <translation> </translation>
    </item>
   ...
  </kernel>
  <graphicsController> … </graphicsController>
 </localisation>
```

Figure 3. Original Language XML file

```
<?xml version="1.0" encoding="UTF-8"?>
 <localisation langID="English">
  <userInterface>
    <item id="56" orig="¿Desea continuar?">
       <translation>Do you want to continue?</translation>
    </item>
       <item id="57" orig="Error: Archivo no encontrado">
       <translation>Error: File not found </translation>
    </item>
   ...
  </userInterface>
  <kernel> ... </kernel>
  <graphicsController> … </graphicsController>
                             </localisation>
```

Figure 4. English Language XML file

Finally, the component "Internationalization Manager" is in charge of detecting XML files in the available languages and thus it offers them as options to the user of the environment. Once the user has select a language, the application dynamically imports the XML file that contains the text strings translated into the selected language and shows the environment in that language.

Conclusion

We have presented three approaches for software internationalization and subsequent localization. We have seen how the use of current programming languages which incorporate XML parsers allows the development of the third strategy, which the one that produces more flexible, adaptable and maintainable applications, in a convenient and easy and straightforward manner with a relatively low cost.

This approach also permits that the work of the developers can be initially kept apart from the linguistic questions and permits to maintain a single version of software. Major changes on the original software can be dealt with in the same way even if there appear new items.

Bibliography

[FSF, 2002] [10] Free Software Foundation (2002), "Online GNU gettext manual" http://www.gnu.org/software/gettext/manual/gettext.html. Visited: March 2006

[Hogan, 2004] [5] Hogan M. J., Ho-Stuart C. & Pham B. (2004), "Key challenges in software internationalisation"

[Huang et al, 2001] [6] Huang E., Hsu J. & Trainor H. (2001), "Unicode enabling for software internationalization" www.symbio-group.com/doc/Symbio%20Whitepaper%20on%20Unicode%20Enabling.pdf. Visited: March 2006

[Huang, 2000] [16] Huang E., Haft R., & Hsu J. (2000), "Developing a Roadmap for Software Internationalization" www.symbio-group.com/doc/Developing%20a%20Roadmap%20for%20Software%20Internationalization.pdf

[LISA, 2002] [13] LISA (2002), "TBX Specification" http://www.lisa.org/standards/tbx. Visited: March 2006

[Lisa, 2004] The Localization Industry Standards Association (2004), "Lisa Industry Primer 2nd Edition".

[LISA, 2005] [12] LISA (2005), "TMX Specification" http://www.lisa.org/standards/tmx/tmx.html. Visited: March, 2006

[Mahemoff et al, 1998] [3] Mahemoff M. J. & Johnston L. J. (1998), "Software Internationalisation: Implications for Requirements Engineering"

[Manemoff et al, 1999] [15] Mahemoff M. J. & Johnston L. J. (1999), "The Planet pattern language for software internationalisation"

[OASIS, 2003] [11] OASIS (2003), "XLIFF 1.1 Specification" http://www.oasis-open.org/comitees/xliff/documents/xliff-specifications.htm. Visited: March 2006

[Stearns, 2002] [4] Stearns B. et al (2002), "e-business Globalization Solution Design Guide: Getting Started" http://www.redbooks.ibm.com/abstracts/sg246851.html?Open. Visited: March 2006

[Tykhomyrov, 2002] [9] Tykhomyrov O. Y. (2002) "Introduction to internationalization programming" The Linux Journal, December, 2002 http://www.linuxjournal.com/article/6176 Visited: March 2006

[W3C, 2005] [2] W3C, "Localization vs Internationalization".http://www.w3c.org/International/questions/qa-i18n. 2005

[Yeo, 2001] [8] Yeo A. W (2001), "Global-software development lifecycle: an exploratory study" Conference on Human Factors in Computing Systems, 2001

Authors' Information

Jesús Cardeñosa – Department of Artificial Intelligence; Universidad Politécnica de Madrid; Madrid 28660, Spain; e-mail: carde@opera.dia.fi.upm.es

Carolina Gallardo – Department of Artificial Intelligence; Universidad Politécnica de Madrid; Madrid 28660, Spain; e-mail: carolina@opera.dia.fi.upm.es

Álvaro Martín – Department of Artificial Intelligence; Universidad Politécnica de Madrid; Madrid 28660, Spain; e-mail: martin@opera.dia.fi.upm.es

EXPERIENCES ON APPLICATION GENERATORS

Hector Garcia, Carlos del Cuvillo, Diego Perez, Borja Lazaro

Abstract: The National Institute for Statistics is the organism responsible for acquiring economical data for governmental statistics purposes. Lisbon agreements establish a framework in which this acquisition process shall be available through Internet, so each survey should be considered as a little software project to be developed and maintained. Considering the great amount of different surveys and all changes produced per year on each make impossible this task. An application generator has been developed to automate this task, taking as a start point the Word or PDF template of a survey, and going through a graphical form designer as all human effort, all HTML, Java classes and Oracle database resources are generated and sent from backoffice to frontoffice servers, reducing the team to carry out the whole set of electronic surveys to two people from non I.T. staff.

Keywords: Software automation, application generators, CASE.

ACM Classification keywords: D2.2. Design Tools and Techniques

Introduction

Complaining Lisbon agreements concerning e-Government, the Spanish National Institute for Statistics (INE) tackles the problem of translating all economical surveys from paper format into web applications. There exist hundreds of different forms, and for a particular survey, more than one version depending on the kind of target organization, so the required effort to create all infrastructure exceeds not only the capacity of I.T. Department, but the budget to carry out the gigantic task. A previous successful experience on metadata processing from INE and the pilot projects on Java application generation from Technical University of Madrid seem a proper combination to afford the trouble.

The idea consists of taking as a start point the current survey forms in Microsoft Word or PDF format, translating these into a tag based format appropriate for both browser representation and automated processing. This creates some kind of template used as a background for the application. Then a user may define the web form over the background painting components using a designer, and establishes properties for the components from those pre-defined in the designer. Finally only translating these definitions into source code is still to be done.

The technology of generated code shall meet the following requirements:

- HTML 4.01, later substituted by XHTML 1.1 by the research team at UPM, for the web user interfaces.
- XForms 1.0, for the definition of validation rules, with the premise to deploy complete surveys in XForms for future use.
- Java servlets, based on action struts architecture and their corresponding beans.
- Hibernate 3 as database connection tier.
- PDF format as receipt of the answered surveys.