

Lessons Learned From Internationalizing a Web Site Accessibility Evaluator

Piotr Rejmer,¹ Michael Cooper,² and Jean Vanderdonck¹

¹Université catholique de Louvain, Louvain-la-Neuve, Belgium,
prejmer@acm.org, vanderdoncktj@acm.org

²Center for Applied Special Technology, CAST, Peabody, MA 01960 USA,
mcooper@cast.org

Abstract

Bobby is a software product that automatically evaluates the accessibility of a Web site according to the guidelines promoted by the W3C. The internationalization of this software poses a series of challenges to support international web sites to evaluate, to incorporate international design and evaluation guidelines, and to produce evaluation reports that can be read by an international reader. This paper reports on lessons learned with this internationalization process.

Introduction

The Center for Applied Special Technology (CAST) is a not for profit organization founded to study and develop ways technology can be used to enable people with disabilities, especially in educational environments, to participate in the mainstream. One of the main products of CAST is Bobby, a software tool that analyzes Web pages for their accessibility to people with disabilities with respect to guidelines maintained by the World Wide Web Consortium (W3C) Web Access Initiative (WAI).

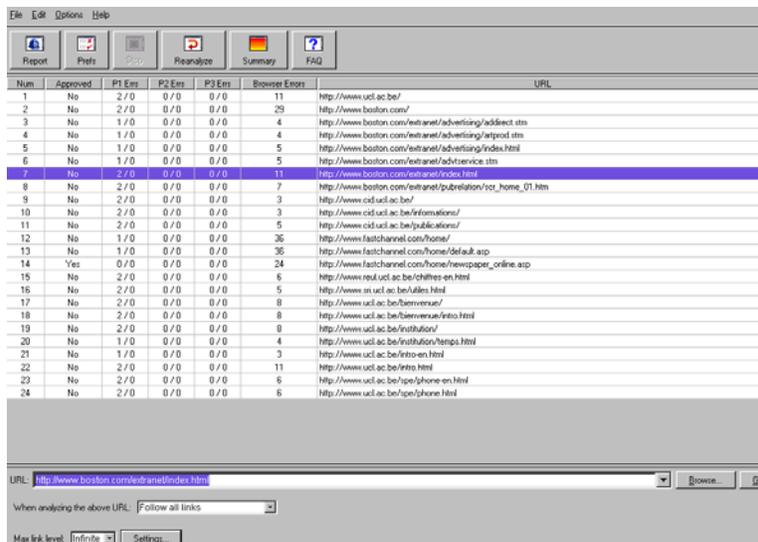
The World Wide Web has no geographic boundaries and has a potential for broad inclusion of individuals—but not of all of them. It often excludes some people from participating in much the same way that a staircase prevents a person in a wheelchair from going in a building's door (Cooper, 1999). Many people, in fact, do not see a web page in the way its designer expected. Individuals “may not be able to see, hear, move or may not be able to process some types of information easily”. They may have a text-only screen or a slow connection. They may have an early version of a browser or a voice browser.

Growing Pains of Designing Global Products

In order to address these issues, the W3C created the WAI, which released the Web Content Accessibility Guidelines (WCAG) in May 1998. These guidelines are intended for all Web content developers (page authors and site designers). In order to help these developers and to promote accessibility, CAST created Bobby to help identify accessibility violations on pages and to provide contextualized instruction in proper implementation of the guidelines.

Bobby is available as a feature of CAST's web site and as a downloadable Java application. It can evaluate web pages on the Internet, and the local version can also examine local files and intranets. The online version by default redisplay the page examined with "Bobby hat" icons to identify the location of high priority accessibility violations, followed by a textual report that describes the location of each violation and links to detailed information about each error type. It is expected that the use of Bobby will help web page authors and designers to take into account accessibility issues so their web pages become usable by the widest audience possible.

Figure 1 graphically depicts the local version of Bobby.



The screenshot shows the offline version of Bobby. At the top, there is a menu bar with 'File', 'Edit', 'Options', and 'Help'. Below the menu bar is a toolbar with icons for 'Report', 'Prefs', 'Reanalyze', 'Summary', and 'FAQ'. The main area contains a table with the following columns: 'Num', 'Approved', 'P1 Err', 'P2 Err', 'P3 Err', 'Browser Errors', and 'URL'. The table lists 24 rows of data, with row 7 highlighted in blue. Below the table, there is a 'URL:' field with the text 'http://www.boston.com/intranet/index.html' and a 'Browse...' button. Below the URL field, there is a dropdown menu for 'When analyzing the above URL:' with 'Follow all links' selected. At the bottom, there is a 'Max link level:' field with 'Infinite' selected and a 'Settings...' button.

Num	Approved	P1 Err	P2 Err	P3 Err	Browser Errors	URL
1	No	2 / 0	0 / 0	0 / 0	11	http://www.ucd.ac.be/
2	No	2 / 0	0 / 0	0 / 0	29	http://www.boston.com/
3	No	1 / 0	0 / 0	0 / 0	4	http://www.boston.com/web/advertising/advertirect.stm
4	No	1 / 0	0 / 0	0 / 0	4	http://www.boston.com/web/advertising/advertirect.stm
5	No	1 / 0	0 / 0	0 / 0	5	http://www.boston.com/web/advertising/index.html
6	No	1 / 0	0 / 0	0 / 0	5	http://www.boston.com/web/advertising/advertservice.stm
7	No	2 / 0	0 / 0	0 / 0	11	http://www.boston.com/intranet/index.html
8	No	2 / 0	0 / 0	0 / 0	7	http://www.boston.com/intranet/pubrelation/loc_home_01.htm
9	No	2 / 0	0 / 0	0 / 0	3	http://www.cid.ucd.ac.be/
10	No	2 / 0	0 / 0	0 / 0	3	http://www.cid.ucd.ac.be/relations/
11	No	2 / 0	0 / 0	0 / 0	5	http://www.cid.ucd.ac.be/publications/
12	No	1 / 0	0 / 0	0 / 0	36	http://www.fastchannel.com/home/
13	No	1 / 0	0 / 0	0 / 0	36	http://www.fastchannel.com/home/default.asp
14	Yes	0 / 0	0 / 0	0 / 0	24	http://www.fastchannel.com/home/newspaper_online.asp
15	No	2 / 0	0 / 0	0 / 0	6	http://www.mil.ucd.ac.be/chef/en.html
16	No	2 / 0	0 / 0	0 / 0	5	http://www.mil.ucd.ac.be/valles.html
17	No	2 / 0	0 / 0	0 / 0	8	http://www.mil.ucd.ac.be/bem/versue/
18	No	2 / 0	0 / 0	0 / 0	8	http://www.mil.ucd.ac.be/bem/versue/intro.html
19	No	2 / 0	0 / 0	0 / 0	8	http://www.mil.ucd.ac.be/institution/
20	No	1 / 0	0 / 0	0 / 0	4	http://www.mil.ucd.ac.be/institution/temps.html
21	No	1 / 0	0 / 0	0 / 0	3	http://www.mil.ucd.ac.be/intro-en.html
22	No	2 / 0	0 / 0	0 / 0	11	http://www.mil.ucd.ac.be/intro.html
23	No	2 / 0	0 / 0	0 / 0	6	http://www.mil.ucd.ac.be/ipe/iphone-en.html
24	No	2 / 0	0 / 0	0 / 0	6	http://www.mil.ucd.ac.be/ipe/iphone.html

Figure 1. Offline version of Bobby

Figure 2 depicts its online version where the user enters the URL of any web page to evaluate.



Figure 2: Online version of Bobby.

Bobby was, however, created for a U.S. English audience, which presents a significant barrier to success in this mission in the international arena. Users around the world have expressed the need of localized versions of the software as their native language may not be English, or their web sites are not located in the United States of America and are not necessarily in English. To address the growing demands of a large international audience and to disseminate its mission to a global market, CAST, with the aid of the UCL, has decided to launch a project of localization and internationalization of Bobby.

Localization here involves taking a product and making it linguistically and culturally appropriate to the target locale (country/region and language) where it will be used. *Internationalization* is the process of generalizing a product so that it can handle multiple languages and cultural conventions without the need for re-design. Internationalization takes place at the level of program design and document level development (Perlman, 2000).

Localizing and internationalizing Bobby means that these processes should be applied to:

1. Bobby, the product itself, for its UI to be appropriate for a wide audience
2. The accessibility guidelines, which may encompass more general guidelines that can be interpreted as culture- or country-independent design guidelines, and/or culture- or country-dependent guidelines

- when the site to be evaluated belongs to a particular culture or country.
3. The evaluation report produced by Bobby subsequently to its evaluation. This report is written in English, which may not mix well with the language of the page that was evaluated, and according to Western culture conventions, which are not necessarily those of the evaluator.

The authors have collaborated on an initial localization of Bobby to a French-speaking audience. This language is spoken in more than 23 countries belonging to three different continents. Belgium itself is especially appropriate because three national official languages are used: French, Dutch, and German.

This paper focuses mainly on localization addressing the first point above, Bobby's UI itself. Addressing this problem does, however, automatically raise a series of questions regarding the two other aspects. The remainder of this paper is structured as follows: we identify a series of problems categories. For each of them, we report in Section II on some significant problems that are systematically presented as a definition of the problem, an illustration of the problem, some ways for solving the problem, and the related guidelines that have been used for this purpose. These guidelines have been either compiled from the existing literature or introduced in the internationalization process itself as the need appeared. Sometimes, guidelines also result from an adaptation of existing guidelines based on different intellectual activities such as abstraction, composition, and generalization. In section III, issues posed by internationalizing the software code are discussed. In section IV, the architecture needs also to be internationalized.

Internationalization of the user interface

Screen Arrangement and Space

Description of the problem. User interfaces are often designed to support one home language, usually English. During the process of adapting a software product to a new culture and/or language, the localization team has to translate the user interface. English is a very compact language, and any translation of text will lead to a new text 50 to 300% longer (especially in the case of single words). Changes in the length of the text will lead to a new arrangement of the interface or will make it impossible to keep the same layout of it (Figure 3).



Figure 3: Lack of space and dynamic layout of the bounding boxes.

Illustration of the problem. The GUI of Bobby has plenty of room to accommodate a longer text in the whole GUI. All the icons in the GUI are a mix of graphics and text. The icons are surrounded by bounding boxes, which makes them look like buttons. The only problem we had is to redefine the size of these boxes which were too small.

Solving the problem. The bounding boxes of the icons including text should be auto-resizable. Bobby has a very “spacey” UI allowing multiple changes or additions of new functions; the auto-resizable option of the bounding boxes will not produce any conflicts or overlaps. A dynamic layout of widgets and room for labels to expand will accelerate the localization process; the team just has to translate the original text without having to modify the layout of the GUI.

Guideline 1:

1. Allow the expansion of displayable text and boundaries boxes in the user interface. Make a dynamic layout of any bounding text boxes.

Keyboard Shortcuts, Mnemonics and Keyboard layouts.

Description of the problem. Keyboard shortcuts are keystroke combinations (consisting of a modifier key and a character key, like Control-Z) that activate a menu item from the keyboard even if the menu for that command is not currently displayed (SUN, 1999). Mnemonics provide a keyboard alternative to the mouse. A mnemonic is an underlined letter in a menu title, menu item, or other interface component. It reminds the user how to activate the equivalent command by pressing the character key that corresponds to the underlined letter.

Shortcuts keys and mnemonics are often dependent of the language and the “computer traditions” of the country (Figure 4). The keys shouldn’t be the same in different localized versions. Conflicts by redundancy are made when you follow the logic rule of the first letter as the character key. It is impossible to use logical associations for the functions of Couper, Copier, Coller (French for Cut, Copy, and Paste). In addition, applications can be used in different countries and across platforms, the keyboards are different

Growing Pains of Designing Global Products

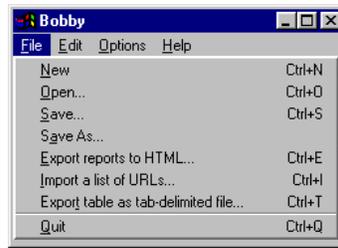


Figure 4: File menu with keyboard shortcuts and mnemonics in Bobby.

with additional keys that are nonexistent on other systems.

Illustration of the problem. In the GUI of Bobby, we use for each function a shortcut key and a mnemonic. The software has to support all major international keyboards. People all over the world will use the English version of Bobby and all the functions of the program must be accessible. Some keys are nonexistent in other countries; for example, the Y key is nonexistent in the Portuguese keyboard. Even keyboards for the “same” language may be different (Figures 5a and 5b).



Figure 5a: Keyboard layout for Belgian French (Kano, 1995)



Figure 5b: Keyboard layout for Canadian French (Kano, 1995)

A shortcut or mnemonic with the Y key would be unusable by Portuguese users. Similar problems exist between mnemonics in English and French (Table 1).

Table 1: Summary of commonly used mnemonics in English and French.

Menu Titles	Menu Items
English: <u>F</u> ile French: <u>F</u> ichier	English: <u>N</u> ew, <u>O</u> pen, <u>C</u> lose, <u>S</u> ave, <u>S</u> ave <u>A</u> s, <u>P</u> age <u>S</u> et <u>u</u> p, <u>P</u> rint, <u>P</u> references, <u>E</u> xit French: <u>N</u> ouveau, <u>O</u> uvrir, <u>F</u> ermer, <u>E</u> nregistrer, <u>E</u> nregistrer sous, <u>M</u> ise en <u>p</u> age, <u>I</u> mprimer, <u>P</u> ropriétés, <u>Q</u> uitter
English: <u>E</u> dit French: <u>E</u> dition	English: <u>U</u> ndo, <u>R</u> edo, <u>C</u> ut, <u>C</u> opy, <u>P</u> aste, <u>F</u> ind, <u>F</u> ind <u>A</u> gain, <u>S</u> elect <u>A</u> ll French: <u>A</u> nnuler frappe, <u>R</u> épéter frappe, <u>C</u> ouper, <u>C</u> opier, <u>C</u> oller, <u>R</u> echercher, <u>S</u> election <u>n</u> ner tout

Solving the problem. The only solution to handle the difference in keyboard layout across platforms or countries is to consult all the major keyboard layouts (Microsoft is providing online a library of all the known layouts) and to spot any potential difference.

Guidelines 2-6:

Since keyboard shortcuts and mnemonics are not always equivalent on different platforms and languages, ensure that any new keyboard shortcuts you have created are compatible with existing shortcuts and mnemonics on all your target platforms.

3. If your software is a cross platform and multilingual application, test the compatibility of the shortcuts and mnemonics on different platforms.
4. Make sure that the characters you have chosen for your shortcuts or mnemonics are available on a maximum of different international keyboards.
5. Internationalization must be addressed along with the concern of keeping the cross-platform portability (Hysel, 1999).
6. Store keyboard shortcuts and mnemonics in resource files, so they can be changed easily for particular language if needed.

Using local conventions for number format and printing formats

Description of the problem. Number representation can vary greatly from one locale to another. An inappropriate formatting changes the sense of the number.

Illustration of the problem. In the U.S. a period is used as a decimal point, but in Europe a comma is used. The thousand separator usage is also different in the continental Europe and the U.S.A. (Table 2). In France and Belgium the legal rule dictates that thousands must be separated by one fourth of a quadratin, also called “fine space”; a quadratin being “a square space with a side length equal to the font size used”.¹ When a fourth of quadratin is not available in the font used, it’s better to use a normal (non-breaking) space than no space at all, or a dot.

Table 2: Part of Bobby report with downloading times expressed with French number formatting.

URL	Size	Time (secs)
http://www.cast.org/bobby/	20,97 K	5,82
http://www.cast.org/bobby/images/dot_clear.gif	0,04 K	0,01

Solving the problem. Localization teams can avoid issues with number formatting by applying typographical conventions, and international guidelines for numerical values and units. They should also ensure that numbers are always output through a locale conversion / formatting function.

Guideline 7:

- | |
|--|
| 7. The software must be able to generate and display numbers using the appropriate local convention (Ishida, 1998) |
|--|

Problems associated with third party software

Description of the problem. Some software developers are using third party add-ons to their software. Developers that wish to add on their core competences use third party solutions.

¹ In a 12 pt font size, thousands should be separated by 3 pt or $3 * 0.3759 \text{ mm} = 1.13 \text{ mm}$.

Illustration of the problem. Bobby after being downloaded is installed on the hard disk of the computer by third party installer software for Java applications: InstallAnywhere from ZeroG. The software uses an English GUI to communicate with the user about installation preferences. This GUI has to be in French to accompany the French localization of Bobby.

Solving the problem. ZeroG offers a French version of InstallAnywhere, it will be probably bundled with the localized French version of Bobby.

Guideline 8:

8. Always check if the third party software you are using and/or adding to your software is localizable or the manufacturer has already made a localized version of it. Ensure that the localized version is the one used or that localization features have been activated.

Text with graphics

Description of the problem. Text can be saved in either as computer-readable text, or as an image in a graphic. Buttons and icons designed for a GUI often mix text and graphic and then save the whole as a graphic file. When in the localization the text must be translated, it can be difficult and time consuming to remove the old text and insert new text.

First illustration of the problem. Access to text saved in a graphic format is difficult, and changes to the text are difficult. Any translation of the text will have an impact on the boundaries of the image and the layout inside of it. The splash screen of Bobby was a pure image with not enough space to allow text expanding. We had to work on an image file. By contrast, the buttons in the GUI of Bobby (Figure 6) were initially designed differently. All the buttons are an association of a graphic file and a text string stored in a resource file. The localization team has only to modify the string inside the resource file to change the text inside the box.



Fig. 6. Initial buttons in the Bobby GUI

Guideline 9:

9. Avoid creating text in a graphic format. Store the text in a resource file. If you do place text in a graphic, ensure that the path to the image is in a resource file so a new one can be substituted.

Second illustration of the problem. Fortunately the original image file of the splash screen was saved in a layered form (*.PSD, Adobe Photoshop), allowing easy changes to the text while keeping the conventions of the image.

Guideline 10:

10. If you have to use a graphic format with text, software allowing layering will make future changes of text easier.

Interface metaphors and creating culturally appropriate graphics

Definition. Computer technology is generally not designed well for humans. Designers of user interface borrow ideas from the real world to bridge the gap. Perceptions of the real world are different from culture to culture and the designers are often unaware of these variances. Some of these differences can result in an interface that is offending or making the interface meaningless.

Solving the problem. The only solution, apart of being aware of cultural biases, is to test the usability of the interface by people from the locale (Ishida, 1998). Even in the case of this localization where the home culture and the localization target culture are so similar, the logo has to be redesigned for the global market. To prevent future color inappropriateness the user will be able to customize the color scheme of the UI. The ability to change colors also benefits users with visual impairment and increases accessibility of Bobby.

It is preferable to review the proposed graphics for international applicability early in the design cycle. Localizing graphics can be a time-consuming process. Although graphics communicate more universally than text, graphical aspects of your software may not have a meaning in many countries. Importantly, some symbols can be offensive in some cultures. Some metaphors also may not apply in all languages. Even if you can create

custom designs for each language, having different images for different languages can confuse users who work with more than one language version.

Illustrating the problem. Figure 7 shows the Bobby logo of a British policeman drawn in a cartoon style. “Bobby the cartoon” has already been determined to be inappropriate for the U.S. audience. Users have pointed that the logo could be perceived as demeaning to individuals with disabilities. The cartoon character for children may encourage the web surfers to view these individual with disabilities in an inappropriate manner. The logo relies on familiarity with “Anglo-Saxon” culture and might be irrelevant for international users.

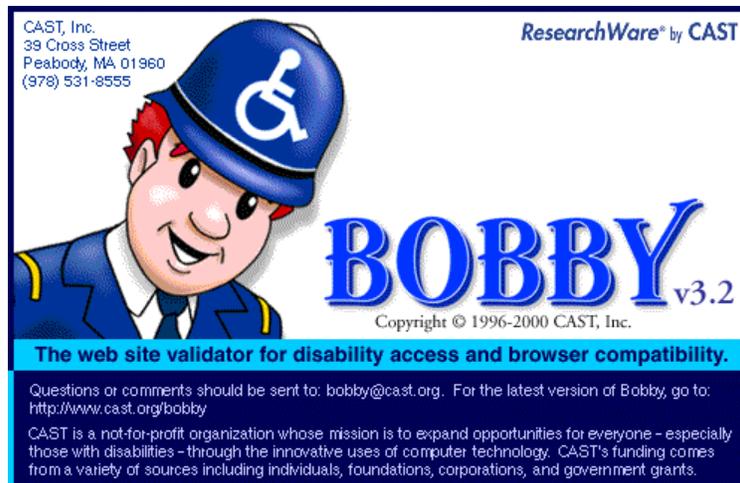


Figure 7: Splash screen and logo of Bobby version 3.2.

Guidelines 11-13:

11. Maintain cultural neutrality in order to make graphics that will be understood globally.
12. The user should be able to customize the color scheme used to conform to other cultural aesthetics (Ishida,1998)
13. Proceed with a usability testing in the locale to detect problematic issues.

Internationalization of the application code

Do you need to internationalize the code? (Sun, 1999)

Before internationalization. You want your application to display messages in the locale language (Here in English, Polish and French). Your code isn't internationalized and you have to write three separate codes (Table 3) and ask for help to translators. Since the translators aren't programmers, you'll have to move the messages out of the source code and into text files that the translators can edit. Also, the program must be flexible enough so that it can display the messages in other languages, but right now you don't know to which countries you will expand.

Table 1. Three separate versions of code, difficult to translate

Separate code for English	Separate code for Polish	Separate code for French
Hello Bobby	Czesc Jacek	Salut Jacques
<pre>public class NotI18N { static public void main(String[] args) { System.out.println("Hello"); System.out.println("Bobby"); } }</pre>	<pre>public class NotI18N { static public void main(String[] args) { System.out.println("Czesc"); System.out.println("Jacek"); } }</pre>	<pre>public class NotI18N { static public void main(String[] args) { System.out.println("Salut") ; System.out.println("Jacque s"); } }</pre>

After internationalization. The source code for the internationalized program is separate from the content (Table 4). The text of the messages is not hard coded but stored in resources bundles according to the locale you targeted (Figure 8).

Composite Strings (Ebben, Marshall, 1999)

A composite string is an error message or other text that is dynamically generated and presented to the user in sentence form. Here is an overly simplistic example (Figure 9):

Table 4. Source code separate from content (internationalized)

Main code – same for every locale	Current locale	Resource files of different locales - MessagesBundle
<pre>import java.util.*; public class I18NSample { static public void main(String[] args) { String language; String country; if (args.length != 2) { language = new String("en"); country = new String("US"); } else { language = new String(args[0]); country = new String(args[1]); } Locale currentLocale; ResourceBundle messages; currentLocale = new Locale(language, country); messages = ResourceBundle.getBundle("MessagesBundle", currentLocale); System.out.println(messages.getString("greetings")); System.out.println(messages.getString("name")); System.out.println(messages.getString("farewell")); } }</pre>	EN,US	<pre>{"dialog_greetings","Hello"}, {"dialog_name","Bobby"}</pre>
	PL,PL	<pre>{"dialog_greetings","Czesc"}, {"dialog_name","Jacek"}</pre>
	FR,BE	<pre>{"dialog_greetings","Salut"}, {"dialog_name","Jacques"}</pre>

```
{"image_alt_smallbobby","Petit logo de Bobby"},
{"image_alt_approved","&Approuve par Bobby"},

{"button_ok","OK"},
{"button_cancel","Annuler"},

{"dialog_about_title","A propos de Bobby"},
{"dialog_options_title","Preferences"},
{"dialog_proxy_title","Configurer le Serveur Proxy"},
{"dialog_quickstart_title","Quickstart"},
{"frame_bobbyfaq_title","Foire Aux Questions sur Bobby"},
```

Figure 8. Sample of a resource file for French locale

```

<% if Err = 400
    errText = "server"
else
    errText = "connection"
end if
%>
U
<P>The <%=errText%> is currently unavailable.</P>
    
```

Figure 9. Example of dynamically generated composite string

This string could not be translated into French. Because “server” is a male noun and “connection” is a female noun, the translation of the term “the” would have to change depending on which condition is met. And because the localizer is given only one opportunity to translate the shell error message, the correct message will be shown for only 50% of the conditions. A better solution would be to have multiple complete error messages that are translated separately.

Alternatively, find a different way of presenting the same information. For example, the following are two ways of telling the user how many mail messages they have (Figs 10a and 10b).

```

MailMsg = [Number of messages in user's mailbox]
<P>You have <%=MailMsg%> message(s) in your inbox</P>
    
```

Figure 10a. Mail message quantity notification, Option 1

Even in U.S. English this is not the clearest communication. However, we can rewrite the code so that it is flexible enough for multiple languages and conveys the same information to the user.

```

MailMsg = [Number of messages in user's mailbox]
<P>Inbox: <%=MailMsg%></P>“
    
```

Fig. 10b. Mail message quantity notification, Option 2

Using resource files

Description of the problem. Localizing requires developers to modify the data and sometimes the code of the program depending on the conception of the software. When the source code is mixed with data it makes any change very difficult because it is necessary to work directly on the source code.

Illustration of the problem. Bobby has a separate program source code and resource files. Resource files are sets of key-value pairs in which a value is substituted every time the key is called. Substituting different resource files allows translation of the content without requiring changes in the source code. The localization team had access to the textual content of nearly all dialog boxes, buttons, and menus without analyzing all the code of the core program.

It is not possible, however, to make changes in the resource files and run the program without recompiling it. This makes the translation process difficult, since it cannot simply be left to a translator but only to a team with both linguistic and engineering skills. Ideally, the UI should be able to be modifiable without recompilation.

Another concern was that the localization work done on Bobby version 3.2 could be lost in a year with the release of Bobby version 4.0. It was important to identify which parts of the resource files were likely to remain constant and focus translation efforts on those parts that could be reused in future versions of the software.

Solving the problem. The core binary executable Bobby file should be completely detached from the user interface contained in resource files. This type of construction will simplify future localizations. Ideally the source program shouldn't include any assumptions on the type of UI that will be used: Command line or Graphical User Interface, each in different possible language versions. The design of Bobby is heading in that direction. The resource files² should include *all* the locale specific data: keyboard shortcuts, displayable text strings, access path to graphic files, colors, mnemonics, etc.

² The locale specific files are renamed based on the ISO 6329 character language code (Perlman, 1999), ex. *BobbyJFCResources* → *bobbyJFCResources_fr*

Guidelines 14-17:

14. Don't combine data and code. Detach the core binary executable file from the user interface. Make the translation a process without recompilation.
15. Store all the information on the user interface in resource files.
16. In order to speed-up the translation, provide comments in the resource files.
17. Develop your software in a modular, extensible and accessible way so future localization can be done more easily.

Other Issues

Character Set

Problems in evaluating web pages that are not in the US-ASCII character set arise from differences among character representation. Many languages require character sets that have more characters than can be defined in an 8-bit encoding. It is necessary to use multiple byte character sets, and different methods are used accomplish this, e.g., some characters sets always use multiple byte characters, while some mix single- and multiple-byte characters and distinguish them by position in the encoding space or by marker characters.

Problems arise when a non-international aware application misinterprets the character represented by a byte. Certain characters have special meaning to the HTML parsing process, such as the angle brackets (<>) that delimit HTML tags. Without attention to internationalization, any byte that matches the byte assignment of one of these characters in the US-ASCII character set is a signal to the tokenizer, regardless of whether it actually represents the character assumed. It is difficult to design a program that can account for these differences and successfully parse any page—there are approximately 100 known character sets in use.

The common solution to this problem is to import the characters into Unicode and base all processing on that character set. The import can only take place, however, if the character set can be determined. There is no algorithm to analyze the byte pattern of a document and reliably detect the character set in which it has been encoded. The HTTP specification defines a way to identify it with the “charset” response header, but this method is rarely used. HTML defines a way to simulate this with a META element, but it requires an initial examination of the document prior to importing and could fail. In addition to these problems, the analyzed document must be re-

exported. Because users can use Bobby in one language and analyze documents in another, the results document must use a character set common to both languages. For many language combinations, the only common character set may be Unicode, which is not yet pervasively supported by document viewers.

Uncertainty avoidance

During the localization process, the Belgian localization team was concerned about the high level of complexity and amount of data on the HTML report produced by Bobby in its U.S. version. Uncertainty avoidance concentrates on the “extent to which people feel threatened by uncertain or unknown situations” (Hofstede, 1991). These feelings of uncertainty are a societal legacy transferred by school education, family environment or state. They are the roots to patterns of collective common behaviors across cultures; it can be translated to “what is different is dangerous”. Belgians have a high index level of 96 points uncertainty avoidance (UA), meaning they emphasize simplicity, clear metaphors, limited choices, and restricted amounts of data within the interface. By contrast, Americans, for whom the interface was originally designed, have an index level of 42 points. The localization has thus required us to step back to work on the internationalization and to include a variety of cultural factors in the redesign process of the application.

Due to limited financial and human resources we have decided to create a single user interface that can be presented in French or English but otherwise is not adaptive (for instance with respect to cultural differences of several French-speaking countries coming from Africa, Canada and Europe). While preserving features and flexibility, we have to meet the needs of the most constraining culture. An example of a change is to give the user the ability to customize the complexity of the interface by using filters on the data that will be presented. The filters can control the depth of the accessibility analysis (Figure 11) limiting it to the most important accessibility criteria or eliminating the display of the HTML code of the issue, e.g. for HTML non-expert users index level of UA. The redesigned interface is intended to be helpful for novice/expert and low/high UA users. A Belgian expert user of Bobby reports finding easier to use at first a simplified version of the report in order to have a rapid overview of the issues and in a second step refine the analysis to study in detail the problems.

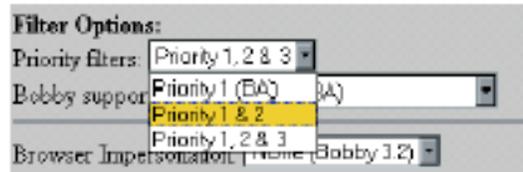


Fig. 11. Depth of analysis

Conclusion

The internationalization and localization are very complex processes. The actual work we have done is only a small step toward the completion of the project. We have identified some problematic issues, some of them are unresolved and probably most of them are still unknown. This experience gave us the opportunity to work in close relation helping us to avoid past, present and future communication problems that has a virtual team located across the Atlantic Ocean and relying largely on e-mail. It has provided foundations for a better future collaboration on the project of localizing Bobby for different markets.

References

- Cooper, M. (1999). *Evaluating accessibility and usability of web pages*, in Proc. of 3rd Int. Conf. on Computer-Aided Design of User Interfaces CADUI'99, Kluwer Academics, Dordrecht, , pp. 21-42.
- Cooper, M. & Rejmer, P. (2001). *Case Study: Localization of an Accessibility Evaluation*, in Proc. of ACM Conf. on Human Factors in Computing Systems CHI'2001, Vol. 2, ACM Press, New York.
- del Galdo, E.M. & Nielsen, J. (1996). *International User Interfaces*, John Wiley & Sons, New York.
- Ebben, S., Marshall, G. (1999) *The localization process: globalizing your code and localizing your site*, msdn online – web workshop, <http://www.microsoft.com>.
- Hysell, D. & Perlman, G. (1999). *Lessons Learned from Internationalizing a Global Resource*, in Proc. of 1st Int. Workshop on Internationalization of Products and Systems IWIPS'99, ACM Press, New York, 1999, pp. 183-192
- Ishida, R. (1998). *Globalization of software and user interfaces*, Xerox,
- Kano, N. (1995). *Developing international software*, Microsoft Press, Redmond.
- Maner, W., *Internationalization of user interfaces*, <web.cs.bgsu.edu/

maner/uiguides>.

- Microsoft (2000). *Windows CE documentation – Application development*, Microsoft Press, Redmond.
- Nielsen, J. (1999). *Designing Web Usability: The Practice of Simplicity*, Indianapolis.
- Perlman, G. (2000). *The FirstSearch User Interface Architecture: Universal Access for any User, in many Languages, on any Platform*, in Proc. of 1st ACM Int. Conf. on Universal Usability CUU'2000, ACM Press, New York.
- Sun Microsystems (1999). *Java Look and Feel Design Guidelines*, Sun Press.