Subtitled interaction: complementary support as an alternative to localization

Giorgos Lepouras\textsuperscript{a,}\textsuperscript{*}, George R.S. Weir\textsuperscript{b}

\textsuperscript{a} Department of Informatics and Telecommunications, University of Athens, Panepistimiopolis, Ktiria Plirrofo, Athens 157-84, Greece

\textsuperscript{b} Department of Information and Computer Sciences, University of Strathclyde, Glasgow G1 1XQ, UK

Received 18 March 2003; accepted 26 June 2003

Abstract

Many computer users face problems in their interaction as a result of the native language employed by the application. The language of the application is often at variance with the native language of its users. This issue is frequently addressed through localization. In turn, localization generates a range of new problems. We propose an alternative to localization that is analogous to cinematic subtitles. This has the potential to reduce the user interaction defects that otherwise arise with localization whilst benefiting users through an additional channel of information in their own language. This paper outlines a prototype implementation and describes our initial evaluation of this approach. We suggest that our complementary ‘subtitles’ promise consistent support for all applications in the user’s computing environment and yield a system that is expandable and much easier to maintain than pre-localized software.

© 2003 Elsevier Ltd. All rights reserved.

1. Introduction

For historical reasons, the majority of interactive systems provide their information in English. However, as the use of computers has spread, the community of end users has gone beyond researchers and scientists to incorporate new user groups. This has resulted in considerable diversity in user needs and social or ethnic backgrounds. In this context, it becomes apparent that many users face problems in their computer interaction as a result of the native language employed
by the software product. The native language of the software is often at variance with the native language of its users.

This problem may be addressed through focus on the software design phase, in order to cater for the needs of local user groups. Among the available methodologies, localization is the predominant approach that is now widely used and advocated.

2. Localisation

Localization is the process of adapting software products to the needs of a local community. This is often coupled with internationalization (Uren et al., 1993), which is the process of isolating language dependant characteristics from the software. If internationalization has taken place, then localization is usually limited to translation of software resources (textual and sometimes graphical components) from English to the local language. Localization has been broadly adopted as the primary methodology for supporting international users, with the presumed benefit that localized software ‘speaks’ the user’s mother tongue. Nevertheless, this is often a putative advantage since software terminology may not be at all familiar, especially to the novice user. Furthermore, localization is known to produce undesirable side effects.

- Disturbing the uniformity and consistency of the users’ working environment. There are two common causes of this problem. The first is the co-existence of localized and non-localized software. In such cases, the user is obliged to learn two sets of terminology: one in the original language and one in the local language. The second cause is that even in localized software there are variations in terminology between different software products, promising further inconsistencies in the users’ working environment (Lepouras and Weir, 1999). Both types of inconsistencies are likely to result in user errors and increases memory and learning load (cf. Shneiderman, 1998).

- Intrusion of foreign cultural values into the environment of localized software. Although localization appears to be solely a language resource issue, a number of other subtle aspects can make a software product appear ‘foreign’ (Russo and Boor, 1993). Ito and Nakakoji (1996) describe a case in which the localization has conflated differences in the cultural context, forcing local users to adopt foreign metaphors. Bourges-Waldegg and Scrivener (1997) suggests that such product adaptation creates ‘cultural impostors’—enabling foreign values to impinge upon a wide range of different cultures.

Localization is less than ideal in environments of users with mixed native languages. In such cases, a localized version should be available for each group of users that needs support. On the one hand, this means a significant extra load on system resources. On the other hand, such an installation often has compatibility problems. Since all localized and non-localized versions use common file names it is
not unusual to have conflicts between them. Even if applications are installed in different directories, they tend to write some files in predetermined directories that cannot be altered. For these files only one version (localized or not) can exist in the system at any time. Such interference is illustrated in Fig. 1 for a context in which a localized (Greek) version of MS-Office 97 was co-installed with an existing English version of MS-Office 97. After installation, the English version was found to have derived some ‘extra’ commands from the Greek version. Such side-effects of localization have been described elsewhere (e.g., by Pemberton, 2000; Weir and Lepouras, 2001).

Although utilization of the user’s mother tongue as the medium of communication is important, one should not underestimate the significance of the presentation technique. Other approaches may offer user support without the limitations faced by localization. Borrowing an example from the film industry, instead of dubbing films to the local language, subtitles can be used. Subtitles provide the necessary information—not always a direct translation—in the viewer’s native language. The main constraints of subtitles in films are limited display space and ‘quick’ dialogues. In consequence, subtitlers often restrict information to a summary of the original dialogue. With computer applications the inverse can be true. The space occupied by the original language resources is usually already limited, making it relatively easy to fit subtitles. Computer users avoid the problem of ‘quick’ dialogues by setting their own pace of interaction.

Since, to some extent, most software applications share a common set of language resources, the role of ‘subtitling’ may be appropriately tackled by dedicated add-on software rather than separately by each individual ‘foreign’ software product. This support facility would monitor user interaction with resident software applications and provide online context-sensitive translation of language resources the user sees on the screen. In contrast to localization, a number of advantages are afforded by this approach. These advantages are summarized in the next section.
3. Advantages of complementary support

3.1. Consistency in the user interface

The proposed approach deals effectively with one of the major disadvantages of localization, i.e., the creation of inconsistencies and the loss of uniformity in the user’s environment. As noted earlier, localization is conventionally applied separately to each software application thereby leading to inconsistencies in the user’s working environment. The end result downgrades the software’s usability, as becomes apparent with the emergence of problems in user–system interaction.

In contrast, the proposed approach treats the local language support problem as a single issue across the user’s entire working environment. Since applications’ user interfaces remain in their original versions, inconsistencies are kept to a minimum. Furthermore, because local support is provided from a separate dedicated application, standardization of terminology is achieved and support provision for all software applications is secured. A further bonus is the prospect of retaining much of the original language context and information.

3.2. Efficient application in multilingual environments

The new approach can be applied especially well in environments where people with different mother tongues co-exist, as in Universities or in multinational companies. In the proposed approach, users can select their preferred support language. Additionally, all support information can be maintained in a single copy for all software applications, conserving system resources and facilitating easier maintenance and updates.

3.3. Intrusion of foreign cultural values

While our alternative to localization has no guarantee of avoiding the intrusion of foreign cultural values, any ‘covert’ aspect is eliminated. Applications retain their original user interface; consequently local users are at least aware of the software’s foreign identity.

4. Design issues

In what follows, we describe our support philosophy and discuss issues related to the development of the standalone application that will monitor users’ interaction and provide on-line context sensitive support. The complimentary support application can exploit the architecture of existing windowing environments that deploy message passing in communication between the operating system and applications. The scheme for this application is depicted in Fig. 2.
The support application comprises three major subsystems.

- Message processing subsystem.
- Support information generation subsystem.
- User interface subsystem.

4.1. Message processing subsystem

The Message Processing Subsystem monitors messages exchanged between applications and the operating system. For efficiency, only messages that relate to user moves are monitored. If a message of interest is found, the Message Processing Subsystem will query the operating system for more information. For example, if the user clicks on a menu then the subsystem will request the menu label. In cases where there are doubts about the translation of a term, the subsystem can retrieve more data, such as the name of the application.

For MS Windows a Message Processing Subsystem can be implemented by exploiting hook functions (Richter, 1995). The hook is a method provided by Windows API that allows for the implementation of a monitoring procedure for application messages. An application can install a hook to monitor messages generated for a certain class of events. When this happens the message will be passed through each corresponding hook function before reaching the application it refers to, as illustrated in the next figure (Fig. 3).

We will outline the procedure for handling messages, used in our implementation in MS Windows 95. Newer versions of Windows may have slightly different API calls, but the procedure should remain the same. In order to install a hook one may use `SetWindowsHookEx` from MS Windows API. The function prototype is as follows.

```c
HHOOK SetWindowsHookEx(int idHook, //the Hook type
                        HOOKPROC lpfn,       // the address of the function
                        HINSTANCE hMod,     // Pointer to the application instances
                        DWORD dwThreadId,   // thread id for which the hook is being installed
);```

Fig. 2. Architecture of support application.
Depending on the hook type, the hook function may monitor messages system-wide or application-wide. For our case the approach of a system-wide hook was adopted. From the range of hooks that can be set, two hook types are of special interest: WH_MOUSE and WH_SYSMSGFILTER. The first type allows the monitoring of messages that relate to mouse events, while the second relates to events that take place in a dialog box, message box, menu or scroll bar.

By calling SetWindowsHookEx with WH_MOUSE hook type one can set a function MouseProc, which will be executed whenever an event relating to a mouse action is received in the message queue. In the case of a system-wide hook, MouseProc has to reside in a dynamic link library (dll). The function template has the following format:

```c
LRESULT CALLBACK MouseProc(
    Int nCode, // hook code
    WPARAM wParam, // message id
    LPARAM lParam // Pointer to a mouse event record (MOUSEHOOKSTRUCT)
);
```

By reading the fields of the mouse event record the MouseProc function can determine the coordinates of the mouse pointer, the location of the pointer (for example inside a window, a menu, etc.) as well as the window (actually a pointer to the window) where the event took place.

The second hook type that can be used is that of WH_SYSMSGFILTER. By calling SetWindowsHookEx function with a WH_SYSMSGFILTER hook type, one can set a function SysMsgProc, which will be executed whenever an event takes place in a dialog box, message box, menu or scroll bar. The function template has the following format:

```c
LRESULT CALLBACK SysMsgProc(
    int nCode, // message type
    WPARAM wParam, // not defined
    LPARAM lParam // address of a structure with data relating to the event
);
```

The nCode parameter can be used to clarify the location where the event took place, while lParam can be used to retrieve information related to the event such as window where it happened and the code of event. Two codes are of special interest: WM_MENUSELECT, which is generated when the user selects a menu or a menu command (left-click on a menu) and WM_COMMAND, which is generated when the user executes a menu command. By reading the fields of the event record, one can determine whether it was a menu item or a submenu selected. In the first case the record will also contain the identification of the item, while on the second case GetSubMenu function can be used to retrieve the identification of the submenu.
With the menu and menu item known, GetMenuItemInfo function can be used to obtain information regarding the menu such as its label. This information, once cleared from special characters such as & and Tab, is being sent to the Support Information Generation Subsystem.

The prototype support system has been implemented to test our hypothesis that this facility will prove at least as effective as localization. Illustrations from this prototype are given in Figs. 4 and 5.

These examples show how this system may provide translation for any application in the user’s environment.

The present version of this support facility provides translation only for terms. This was considered sufficient for assessment purposes and an experiment was set up to test the use of this alternative to localization. Our experiment is described in the next section.
4.2. Support generation subsystem

The Support Information Generation Subsystem accepts three types of input: terms, messages and free text. Terms can be translated easily by means of a terminology database. Message translation has two options: the first is to use a database of messages along with a parser function to extract variables such as filenames, from the messages, and the second to use a machine translation system. The first option is much easier to implement but more difficult to maintain than the second. Finally, for the translation of free text, a machine translation system is the only option. Since our research focus lies in the concepts of user support, rather than translation techniques, we are content to adopt any compliant machine translation system that affords accuracy in the translated text. This is a practical proposition where the 'controlled' language of help texts or system documentation is the target.

4.3. User interface subsystem

Translated text either in the form of terms, messages or free text is passed to the User Interface Subsystem to be presented to the user. Depending on the information type, a number of presentation techniques can be used.

For terms or small phrases, a tool-tip style window can be used to display online translations. The support afforded by this method is tightly coupled to the object the user points to with the mouse. However, problems may arise in cases where the support corresponds to information that is already being displayed in a tool-tip window. In such a case the support information can be either overlaid on top of the original tool-tip window (as is often the case when subtitling films already subtitled) or displayed in a tool-tip window beneath the original.

Alternatively, the translated information may be presented in a floating window. This offers the potential of more display space (to present for example the translation of larger text sections or even the translation of a complete menu instead of just the selected item) at the expense of being tightly coupled with the object of interaction.

Fig. 5. Complementary support for WordPad™.
5. Experimental Evaluation

5.1. Initial considerations

The objective of the experiment was to contrast the effectiveness of our complimentary support approach in comparison to (a) localization, and (b) the absence of any local language support (i.e. the use of software in the original language).

The main hypothesis under test was that user support provided by the proposed approach was at least as effective as that of localization. As a target application for this evaluation, we selected MS Word '95 since this was available both in original and localized versions. To minimize external influences, we sought participants with little or no prior experience with the software application (or even with computers).

5.2. Scenario description

On the basis of these considerations, we formed a scenario for the experiment. First of all, subjects should be novice users of word-processing applications with a minimum experience of computers. To acquaint themselves with the applications they would be given three 90 min training sessions conducted over a period of 1 month. The first session would cover operating system basics, while the other two would focus on the core features of word-processing. Subjects would be divided randomly into three groups. The first group would be trained using the original version of the software application with the complementary support. The second group would be trained using the original version of the software without any other type of local language support, while the third group would use the localized version of the software.

In each training session, regardless of the subject’s group, the instructor would also seek to familiarize participants with relevant terminology, including the correspondence between original and local language terms. By this means, we sought to balance participants’ exposure to terminology.

Upon completing the training sessions, subjects would engage in the assessment. This would comprise three phases. In the first phase, participants would format an unformatted text, in a predetermined amount of time and to match a given sample. At the end of the time period subjects would save their formatted text to disk. Since tasks carried out in this phase would be unstructured, i.e., subjects could execute whatever steps they felt necessary to reach the desired format, logging of their actions was not performed.

Logging of user actions was appropriate during the second phase of the assessment, where participants would have to complete a specific set of word-processing tasks. Through log file analysis, measurable criteria, such as speed of task execution and rate of errors, could be extracted. To obtain a log of the user actions, a monitoring application was implemented. This records user–system interaction and outputs a file containing action descriptions and associated timestamps.
Although this phase was structured, carrying out tasks in the wrong order, such as first formatting the font and then hiding the toolbar, instead of the reverse, would not count as an error. On the other hand, if a subject executed an action that was not relevant to the scenario, this would count as an error.

In the third phase of the experiment, participants would complete a questionnaire spanning profile (age, knowledge of English language, computer experience, etc.), subjects’ satisfaction (ease of use, command ease of understanding, etc.) and questions rating the subjects’ ability to remember terminology and identify commands by their menu location.

A total of 21 participants took part in the assessment phase, and these were divided into three groups of seven subjects. The number of subjects per group is close to the minimum that can give results from a statistical analysis. However, it was felt that such an assessment, even with a small number of participants, could provide valuable insights on the efficacy of the proposed method.

Participants were high school graduates who had recently gained admission to the Department of Mathematics. The subjects, who were briefed on the nature of the experiment and the methods of acquiring data, had no prior experience of word-processing software and little previous knowledge of computers.

5.3. Assessment results

For the statistical analysis of results the Kruskal–Wallis non-parametric test was used. This tests the statistical significance between three or more independent samples. If the value of $P$ (the possibility of the populations having the same mean values and samples of these populations having different mean values) is less than 0.05, this rules out the assumption that these differences are random. In other words, such a result signifies that at least one group differs significantly from the others. To identify which of the groups produces this result one can use Dunn’s test.

Table 1 summarizes the results from the first phase of the experiment. This tabulates the number of successfully completed formatting tasks (within a maximum of ten tasks) carried out by subjects.

Users from the first group (complementary support) managed to complete a greater number of tasks in the given time than users from the other two groups. All users completed at least 9 out of 10 formatting tasks, while only three users of Group B completed at least 9 tasks and only 2 users of Group C completed more than 9 tasks. The value of $P$ for Table 1 is 0.1234, which is considered non-significant.

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Phase 1 results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of completed formatting tasks</td>
<td>10</td>
</tr>
<tr>
<td>Group A (complementary support)</td>
<td>4</td>
</tr>
<tr>
<td>Group B (original version)</td>
<td>3</td>
</tr>
<tr>
<td>Group C (localized version)</td>
<td>2</td>
</tr>
</tbody>
</table>
To ease the collection and analysis of the second stage of the experiment a monitoring tool was developed which logged user dialogues with the word-processing software. The tool recorded elapsed time and user action, and stored this data in a file. At the end of the experiment, data was collected from each participant’s computer. Data was then imported to a spreadsheet. This simplified the extraction of task times as well as the number of errors. We regard all actions that did not contribute to the completion of a task or failure to complete a task as an error. Completing tasks in a different order was not thought to be an error so long as the end result was the same. Use of the ‘Undo’ operation was not counted as an error.

Our results from the second phase of the experiment, along with the associated \( P \) value, are indicated in Table 2.

With tasks for which the value of \( P \) was less than 0.05, Dunn’s test was employed to help identify which group exhibited a statistically significant difference. The results from this analysis are summarized in Table 3. Values less than 0.05, indicate a difference between the corresponding groups that is statistically significant.

From the statistical analysis of our second phase results, the following conclusions can be drawn.

- There is a statistically significant difference between the first and the second user groups, in tasks that are less often carried out. For tasks carried out often, like ‘Save’ and ‘Exit’, the difference is not statistically significant (although this may also be attributed to the small number of subjects per group).
There is a statistically significant difference in task execution time between the first and the third user groups for those commands that are not commonly used.

There is no statistically significant difference in task execution time between the second and the third user group, although one would expect that users with access to the localized application would have done better than users of the original version of the software. As previously, this could be attributed to the small number of subjects per group.

Analysis of the users’ questionnaires, collected in the third phase of the experiment, provided information regarding the subjects’ profile and opinion as well as terminology and command retention. We report only those results that are of some significance. For each of the questions presented we provide a figure illustrating the subject’s response to the question as well as the result of Kruskal–Wallis statistical significance test (to identify statistically significant differences between the answers of each user group). In order to quantify subjects’ responses we assigned a number to each possible answer (for example 5 to Very Good, 1 to Don’t Speak). The resulting matrix was the basis for statistical analysis.

The first two questions concerned subjects’ background. Most subjects had an average knowledge of English, which was rather anticipated since in Greece English is the foreign language most commonly taught in schools. The value of $P$ is 0.7050, which is considered not significant; indicating that the distribution of subjects to groups in relation to their knowledge of English was rather consistent (Fig. 6).

As expected, most subjects had little overall experience with computers. The value of $P$ is 0.8251, which is considered not significant, indicates that the distribution of subjects to groups in relation to their experience with computers was also consistent (Fig. 7).

The next two figures present the subjects’ opinions in regard to the comprehensibility of commands and to the functionality of software they used. The value of $P$ for the results of the first question is 0.1738 (Fig. 8), which is considered not significant. The value of $P$ for the results of the second question (Fig. 9) is 0.2460, which is considered not significant.

<table>
<thead>
<tr>
<th>Task description</th>
<th>Dunn’s test</th>
<th>Dunn’s test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A vs. B</td>
<td>A vs. C</td>
</tr>
<tr>
<td>Hide toolbar</td>
<td>&lt;0.01</td>
<td>&gt;0.05</td>
</tr>
<tr>
<td>Format font</td>
<td>&lt;0.01</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td>Hide ruler</td>
<td>&lt;0.01</td>
<td>&gt;0.05</td>
</tr>
<tr>
<td>Insert table</td>
<td>&lt;0.05</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>Copy</td>
<td>&gt;0.05</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td>Total time</td>
<td>&lt;0.05</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td>Number of errors</td>
<td>&lt;0.01</td>
<td>&lt;0.05</td>
</tr>
</tbody>
</table>
Fig. 6. Subjects’ knowledge of English.

Fig. 7. Subjects’ computer experience.

Fig. 8. Rating of command comprehensibility.
Furthermore, it was found that users with supplementary support could remember the corresponding Greek or English term and the command from the position of the menu more often than users from the other groups. The value of $P$ is 0.0034, which is considered significant (Fig. 10).

Applying Dunn’s multiple comparisons test to this data shows that the value of $P$ for Group A vs. Group B is less than 0.05—considered significant; for Group A vs. Group C $P$ is less than 0.01—considered significant, while for Group B vs. Group C is more than 0.05 and is considered not significant. Subjects of the first group were able to remember a larger percentage of terminology (pairs of original and local language terms) as depicted in Fig. 10. This observation indicates that users of the first group will be able to cope better in mixed-language environments.
In another question, subjects were shown pictures of two menus from the software they had used (Groups A and B from the original language version, Group C from the localized version) with the name of a command hidden, and they were asked to recall the missing command. The results from this question are shown in Fig. 11. The value of $P$ is 0.4113, considered not significant.

6. Conclusions

The alternative to localization described in this paper is analogous to cinematic subtitles. This has the potential to reduce some of the user interaction defects that otherwise arise with localization. Moreover, users benefit from an additional channel of information in their own language. This native language content serves as complementary support to the original application.

By exploiting programming hooks, the proposed system can monitor user actions and provide online, context sensitive support. The Application Programming Interface offers the necessary resources for implementing this type of support. While testing the support application with newer versions of the operating system and with applications compliant with these versions it was found that in some cases the support failed to recognize the messages exchanged and provide the corresponding support. This type of reaction was somehow anticipated, since with every new version of the operating system, new features and functions supersede the old ones, which become obsolete. To this end, the support application has to adapt to changes in the Application Programming Interface, in order to be able to offer support for newer applications. On the other hand, for older applications backward compatibility in the programming interface ensures that they will be supported.

This strategy affords consistent support for all applications in the user’s computing environment. The system is expandable and much easier to maintain than pre-localized software. Popup support for applications is ‘automatic’ even for
newly installed programs (cf. Figs. 12 and 13). We also believe that this system helps users learn and retain terminology.

The work presented in this paper is part of our on-going research on second language support. The next phase includes extending the support mechanism to embrace other means of presentation (supplementing tooltips with floating windows, status bar output and even native language voice output). In due course, these developments will also be subject to evaluation and field trials. The next phase will also investigate the implementation issues for developing this type of support for other operating systems, and especially that of Linux.
References