International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:3, 2007

Automatic Rearrangement of Localized Graphical User Interface

Ágoston Winkler, and Sándor Juhász

***Abstract*—**The localization of software products is essential for reaching the users of the international market. An important task for this is the translation of the user interface into local national languages. As graphical interfaces are usually optimized for the size of the texts in the original language, after the translation certain user controls (e.g. text labels and buttons in dialogs) may grow in such a manner that they slip above each other. This not only causes an unpleasant appearance but also makes the use of the program more difficult (or even impossible) which implies that the arrangement of the controls must be corrected subsequently. The correction should preserve the original structure of the interface (e.g. the relation of logically coherent controls), furthermore, it is important to keep the nicely proportioned design: the formation of large empty areas should be avoided. This paper describes an algorithm that automatically rearranges the controls of a graphical user interface based on the principles above. The algorithm has been implemented and integrated into a translation support system and reached results pleasant for the human eye in most test cases.

***Keywords*—**Graphical user interface, GUI, natural languages, software localization, translation support systems.

I. Introduction

THE world-wide improvement of the information infrastructure has caused an exponential growth in the number of computer users and significantly modified their composition and computer using habits as well. Whereas computer programs were basically used by researchers and experts in the past decades (who generally spoke foreign languages and used computers typically in the field of their own specialties), today's users represent a much wider range of the population and use computers at different places (at home, at their working place, at school or even during traveling), for different purposes (work, education, entertainment, searching for practical information etc.). However it is a serious problem for many of the new users that certain computer programs only have an interface in a foreign language. This problem has been recognized by the software development companies as well thus they have realized that the only way to reach more users on the international market is adapting their software products to the local requirements, i.e. localizing them.

Manuscript received August 30, 2007. The authors are with the Department of Automation and Applied Informatics, Budapest University of Technology and Economics, 1111 Budapest, Goldmann György tér 3., Hungary (e-mail: agoston.winkler@aut.bme.hu, sandor.juhasz@aut.bme.hu, phone: +36 (1) 463-1648, fax: +36 (1) 463-2871).

Localization is a complex process: beside the translation of the text elements of the user interface it is often required to modify some functions of the program as different countries have different specialties, traditions, standards and legislation. (As an example, an accounting program needs to comply with the laws and regulations of the specific country.) However the most important step is still the translation of the user interface. Translation can be supported and even automated very efficiently [1], [2], [3], nevertheless the localization process is not completed with this step.

It is quite a serious problem that the size and the arrangement of the controls (e.g. text labels, buttons, list boxes etc.) is usually fitted to the extension of the texts in the original language. As these sizes are subject of changing during the translation, the localized interface may become distorted [4]. If controls only shrink, the problem is not significant as the original layout remains suitable for the new language as well. On the other hand, the growth of sizes may cause that certain controls slip above each other which results an unpleasant appearance. Furthermore, it may make the use of the program more difficult or even impossible.

Unfortunately, researches have found that the length of translated texts is much more likely to increase than to decrease. This phenomenon can be explained with more reasons. The first one is the so-called "explicitation" that means the process of adding information to the target text explicitly that is only implicit in the source text [5], [6]. Explicitation can be obligatory when the grammar of the target language forces the addition and voluntary when the aim is to make the text easier to understand [6]. As an example, the English version of Microsoft Internet Explorer asks the user to "Click Custom level." It is not explicitly told that it is a *button* that should be clicked but it is not even required in this language. On the contrary, certain languages (like Hungarian) need to specify this piece of information as well, for better comprehension. The English version does not either contain the subjective of the sentence explicitly. In some languages (for instance in German) this is also obligatory.

There are other effects that may increase the length of translated texts. For instance, the different average word length results that sentences will be 19% longer after translating from English to French [7] and 30% longer after translating to Spanish [8].

The problem caused by the growth of text sizes can be prevented by proper foresight. User interface designers may

measure the controls so that they fit even the longest translation. Alternatively, modern design techniques can be used to make the product "ready for localization" [9], [10]. Unfortunately, the international distribution of the product is often not planned at the beginning of the development process causing that developers do not use these techniques. As result the problem appears only when the final product is ready.

In this case, the arrangement of the controls must be corrected subsequently. The correction should preserve the original structure of the interface (e.g. the relation of logically coherent controls that are originally in one row or one column should remain the same), furthermore, it is important to keep the nicely proportioned design of the interface (the formation of large empty areas should be avoided). The transformation can be processed with human contribution, however this solution is quite time-consuming and does not ensure the desirable precision by all means.

This paper presents a new algorithm that automatically rearranges the controls of a graphical user interface based on the principles above. The organization of the paper is as follows. Section II provides an overview of translation support systems and places the algorithm in the localization process. Section III presents the structure of the input data and the basic scheme of the algorithm, and then it describes each step in details. Section IV introduces the achieved results whereas Section V summarizes the work and mentions some possible ways of further improvement.

II. Translation Support Systems

As already mentioned, thanks to the repetitions and frequently occurring expressions, user interface translation can be automated very efficiently [1], [2], [3]. Fig. 1 presents the process of localization.

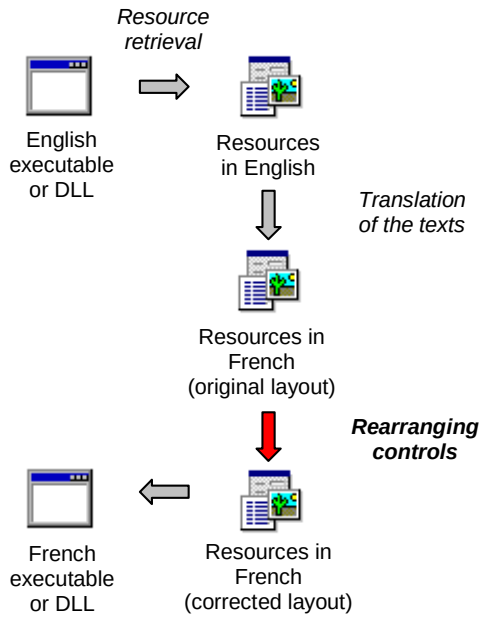


Fig. 1 Steps of translating resources in a program file

Translation support systems are able to retrieve the resources (dialogs, string tables etc.) from the executable program files and libraries. Then they help users with translation memories, dictionaries, spell checker modules and interactive interfaces to do the translation as automatically as possible. Finally, the translated versions of the resources are saved in a copy of the original program file. As Fig. 1 shows, the rearrangement of the dialog controls should be done after the translation. The algorithm that is presented in the next section serves the automation of this step.

III. The Rearrangement Algorithm

The input of the algorithm is a list of the controls that contains their original coordinates and sizes as well as their new sizes after the translation. The type of the controls (text label, button, list box etc.) is not exploited by the algorithm: although their distinction would be useful in some special cases, the main goal is to work universally, with all kinds of controls (even with those of the future). The output of the algorithm is a list of the same type, of course with the modified coordinates (and sizes).

The algorithm can be split into three main phases which is shown in Fig. 2 [11]. As the input data contains no information about the hierarchy of the controls, it must be determined by the algorithm. This is the most important task of the *pre-processing phase*. Furthermore, a list must be created that contains the control pairs that overlap each other in the original layout so that they can be handled specially later. The next phase is the *rearrangement* itself that is executed level by level, starting from the bottom of the control hierarchy. Finally, the *post-processing phase* contains some optional fine-tuning steps.

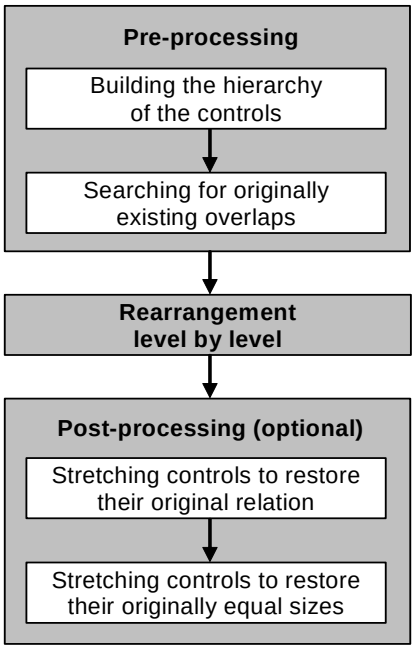
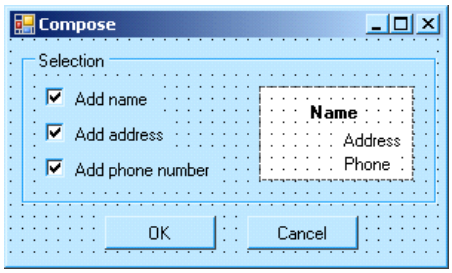


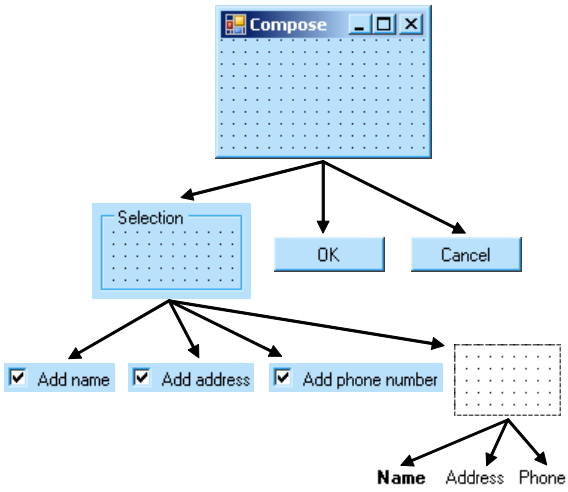
Fig. 2 Phases of the rearrangement algorithm

A. Pre-Processing

As mentioned before, the first and most important task of the pre-processing phase is creating the hierarchy of the controls as the input of the algorithm does not contain it explicitly. This is required for being able to execute the rearrangement algorithm level by level. The hierarchy tree can be built by using the coordinates and the sizes of the controls. In order to eliminate the incidental imprecision of the design, a tolerance value is used during the determination of the parent-child relations. Fig. 3 shows a sample dialog and the hierarchy tree of its controls.



(a)

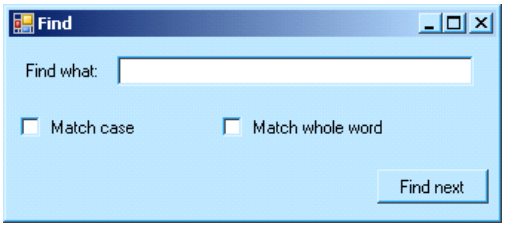


(b)

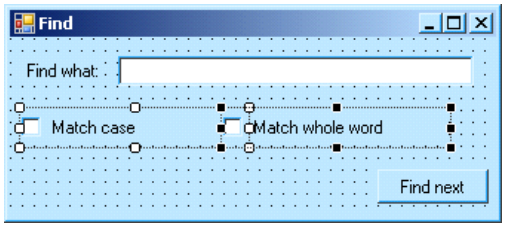
Fig. 3 (a) A sample dialog (b) The hierarchy tree of its controls

One of the basic principles of the rearrangement algorithm is that the creation of overlaps due to the translation must be avoided, i.e. controls that are originally isolated, should not slip above each other. However practice shows that sometimes the original layout contains overlaps as well. This may be caused by the impreciseness of the designer (see Fig. 4) but it can be intentional as well, usually when using standard controls (e.g. panels) for reaching special graphical effects (e.g. shadow, see Fig. 5).

Whereas the creation of new overlaps must be avoided, the originally existing ones should be handled specially. This implies that original overlaps should be remembered, and after the rearrangement they must be taken in consideration.



(a)



(b)

Fig. 4 (a) Checkbox controls seem to be isolated when the program is running (b) However, in design view it can be seen that they actually overlap each other because of the imprecision of the design

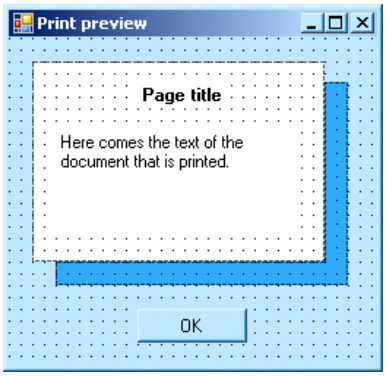


Fig. 5 Reaching a shadow effect by using overlapping panel controls

B. Rearrangement Level by Level

The basic rule (Fig. 6) of the rearrangement is quite simple. The algorithm iterates all the controls from top to bottom, from left to right, starting from the top left corner (the actual one is referred as the *primary control*). It calculates the growth of each control and examines whether it causes any new overlaps. If so, it moves all the other (so called *secondary*) controls that are right from the right edge of the primary control to the right (in case of horizontal growth), whereas it moves the secondary ones that are below the bottom of the primary control downwards (in case of vertical growth), in such a measure that the formation of a new overlap is just avoided. (A minimal distance to be kept between controls can be specified as well.) The collective replacement of the controls ensures that logical groups will not be split up.

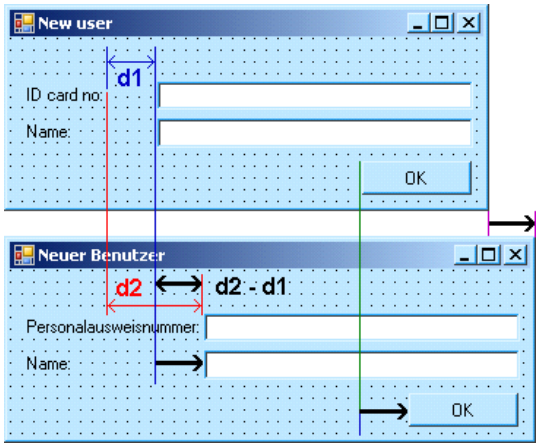


Fig. 6 The basic principle of the algorithm: the primary control, text label "ID card no" can grow by *d1* without causing a new overlap; if its growth is *d2*, all secondary controls will be moved to the right by *d2* - *d1*

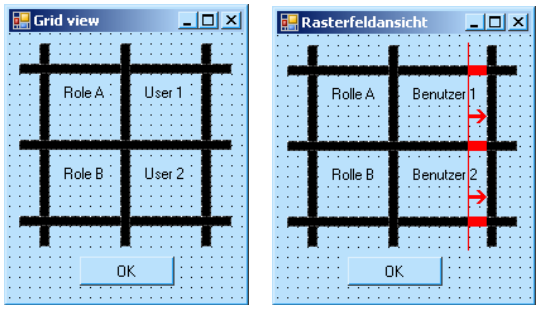
As already mentioned, controls that originally overlap each other need to be handled specially. If the overlap is intentional (like in Fig. 5), the aim is to preserve the original relation of the controls. This implies that secondary controls overlapping originally and being right from the left edge (or below the top) of the primary control should be moved the same way as in general cases, in accordance with the growth of the primary control. Furthermore, this behaviour is suitable even if the original overlap is the result of an imprecise design (like in Fig. 4). This case can be considered as having no original overlap between the controls but no additional space on the right side (or under) the primary control either, i.e. the bottom right corner of the primary control and the top left corner of the secondary one move together.

As mentioned before, the rearrangement algorithm described above is executed hierarchically level by level, from bottom to top. While going up from smaller groups towards the complete form, the composite objects of the previous level are treated as single, monolithic controls the size of which was determined by the algorithm described above.

C. Post-Processing

Although tests have shown that the presented algorithm reaches quite good results despite its simplicity, it is possible to provide even better results by handling some special cases.

One of these areas is the use of standard controls for reaching special graphical effects like in Fig. 5. However, whereas the dialog presented in Fig. 5 contains only two overlapping controls, it is frequent that more controls are effected. The standard algorithm presented in subsection III.B moves only the secondary controls that are below or right from the primary control. The position of the other secondary controls is already fixed and should not be changed again as it could result an infinite loop. Fig. 7 shows an example, a grid composed by panel controls.



(a) (b)

Fig. 7 (a) Reaching a special grid effect by using 6 overlapping panel controls (b) Restoring the original relation of the controls by stretching the horizontal panels

If the text labels "User 1" and "User 2" grow, moving the right side panel would cause that the horizontal panels are moved, too. However these panels would "drag" the left and the central vertical panels as well that would slip above the text label. To avoid the formation of the new overlap, the text labels and the right panel would be moved again to the right, starting the whole procedure again. In such cases it can be a suitable solution not to move but to stretch the originally overlapping secondary controls that are above or left from the primary control. This restores the original relation of the controls but does not require subsequent operations. However as this behaviour may be undesirable because of changing the ratio of control sizes, it is only an optional parameter of the algorithm.

An other function that may be used particularly for aesthetic improvements is the equalization of the sizes of controls that have the same horizontal or vertical starting position and originally had the same height or width as well. Fig. 8 shows an example. This feature is optional as well as it is not suitable in all cases: equal positions and sizes may be the result of a simple coincidence. The decision would be easier if the type of the controls were known as it is more likely to have a real logical group if the type of the elements is the same. However as it was already mentioned, this piece of information is not exploited by the algorithm.

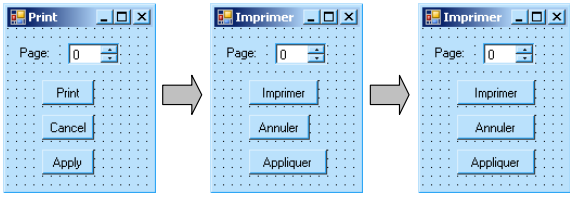


Fig. 8 Restoring originally equal button sizes that were grown in a different measure during the translation

IV. Results

The rearrangement takes $O(n^2)$ steps without the optional post-processing features (described in subsection III.C) and $O(n^3)$ including them (n is the number of the user controls).

This implies that the algorithm can be used in practice as well [11].

The algorithm has been implemented and tested with real life examples. The total processing time of a test file containing 177 dialogs was 5.22 seconds that is quite a good result (C# implementation, Microsoft Windows XP operating system, .NET framework 2.0, Intel Core ™ 2 CPU 6400 2,13 GHz, 2 GB RAM, 200 GB HDD).

Fig. 9 shows some detailed experimental results. Controls were added to a simple dialog (containing only one hierarchy level) one by one and the processing time of the dialog was measured (with and without the post-processing steps). The measurement has nicely justified the theoretically predicted $O(n^2)$ and $O(n^3)$ complexity of the algorithm.

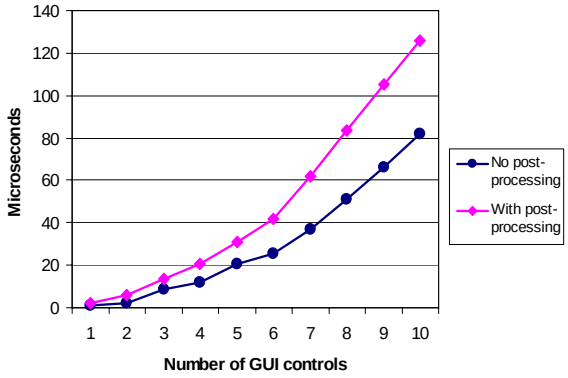


Fig. 9 Processing time of a simple dialog containing different number of controls, with and without post-processing steps

The rearranged dialogs were reviewed by a human tester who found no significant errors.

V. Conclusion

This paper presented a new algorithm that supports software localization by automatically rearranging the controls of a graphical user interface after its translation in order to restore its usability by keeping its nicely proportioned design.

Next to the basic tasks we also handle some special cases like overlapping controls and visual effects achieved by using standard controls. The algorithm is kept relatively simple, it has a cubic complexity, but because of the low number (few hundreds at maximum) of graphical elements in a single form, it can be used interactively as processing of a complete form does not take more than a few hundredths of second.

We considered it as an advantage that the algorithm does not exploit the knowledge of the type of the controls, as this allows a universal application of it. On the other hand, in some special cases it would be worth using this piece of information (of course, the basic algorithm should work without it as well). A typical application would be the post-processing described in subsection III.C to decide which modifications should be applied. However this improvement would affect only the aesthetics of the transformation as its basic goal is already achieved by the present algorithm.

References

[1] Passolo homepage, http://www.passolo.com/
[2] Trados homepage, http://www.trados.com/
[3] MemoQ homepage, http://www.kilgray.com/kilgray/products/memoq?locale=en
[4] Gary F. Simons, John V. Thomson, "Multilingual data processing in the CELLAR environment," in *Linguistic Databases*, Groningen, 1995, pp. 203-234.
[5] Silvia Hansen-Schirra, "Linguistic enrichment and exploitation of the Translational English Corpus," in *Proceedings of the Corpus Linguistics Conference*, Lancaster, 2003, pp. 288-297.
[6] Ana Frankenberg-Garcia, "Are translations longer than source texts? A corpus-based study of explicitation," in *Third International CULT (Corpus Use and Learning to Translate) Conference*, Barcelona, 2004, pp. 2-9.
[7] Aletta Grisay, "Translation procedures in OECD / PISA 2000 international assessment," in *Language Testing*, SAGE Publications, 2003. Avaible: http://ltj.sagepub. com/cgi/content/abstract/20/2/225
[8] Diana Díaz Montón, "The Video Game Translator Wishlist," in *Gamasutra*, 15 June 2005. Avaible: http://www.gamasutra.com/features/20050615/ monton_01.shtml
[9] Lingobit, "Preparing User Interface for Localization." Available: http://www.lingobit.com/solutions/preparing_gui.html
[10] "Creating a Layout That Adjusts Proportion for Localization," in *Microsoft Developer Network*, http://msdn2.microsoft.com/en-us/library/7k9fa71y (vs.80).aspx
[11] Ágoston Winkler, "A New Auto-Layout Algorithm for GUI Localization," in *Automation and Applied Computer Science Workshop 2007 (AACS'07)*, Budapest, 2007, pp. 295-300.