Proc. of the Intl. Conf. on Advances in Computing and Information Technology-- ACIT 2014

ISBN: 978-981-07-8859-9

Software Internationalization: Incorporating Users' Translations

Jauhar Ali

***Abstract*—Internationalized software, which support localization by allowing users to choose the user interface language of their choice, has a lot wider audience than a single-language software. Providing resource files for multiple languages is expensive and not scalable for large number of languages. This limits the availability of software products to only those communities who speak major languages. In this study, we investigate the idea of allowing users to create user interface translation of software and share them with other users. We believe that such an approach is scalable as it does not incur any extra cost.**

***Keywords*—internationalization, localization, user interface**

I. Introduction

The user interface of any software is based on a particular natural language. Well-internationalized software supports other languages by providing localized versions of the software, each targeting the language of a particular community. However, such localization is very expensive and not scalable [1]. Software producing companies target only few major communities for the localized versions of the software. No software producing company can afford providing localized versions for all communities. When the population size of a spoken language is small, it is less likely that a software will use that language for its user interface. Also, for such a small market there is not much incentive for software producing companies to make effort to offer a localized version of the software for that language. This results in the limited availability of niche software products in minorities' languages [1], [2].

Jauhar Ali
College of Engineering and Computer Science
Abu Dhabi University
PO Box 59911, Abu Dhabi
United Arab Emirates

The main technique for internationalization of software is to create resource files for each target language. The resource files contain a list of key-value pairs, where all values are translations of the keywords to be displayed in the user interface. For each target language, a separate resource file needs to be created and packaged with the software. Users can select the language of the interface to be one for which a resource fie is created and packaged by the software developer. The cost of creating resource files is the main obstacle in providing localized software for languages of minor communities. Crowdsourcing [3], a practice of getting a task performed by a group of people that offer their services, has the potential to create user interface translations at a very low cost.

This paper describes a method through which users can create translations of a software's user interface and share them with other users. Users can also modify translations which are shared by others. The resulting translations may not be perfect as they are done by end users, but they are produced at no cost and they help in making the software available to minor communities. The approach is especially beneficial for creating translations of educational and entertaining software products for kids by their parents or guardians.

II. Related Work

Many books have been written on software internationalization and localization [4], [5], [6]. Some textbooks on programming [7] also covers the topic because of its importance. Khadam and Vanderdonckt [8] have reported a method for solving the problem of Arabic and Hebrew languages reading from right to left instead of the left to right of Western languages. Peng, Yang and Zhu [9] addresses the need for reengineering existing code that has been written in ANSI format to the preferred Unicode for ease of internationalization. The requirement to reengineer existing software is also addressed by Wang, et al. [10] who describe a technique for finding the "need-to-translate constant strings".

Rößling [11] has designed a Java Package that recreates some of the most commonly used Java Swing (graphical

Proc. of the Intl. Conf. on Advances in Computing and Information Technology-- ACIT 2014

ISBN: 978-981-07-8859-9

user interface) components to deal with internationalization issues such as changing text, formatting numbers, and changing icons. The idea of changing icons is a good contribution for localization but it does not offer a full solution to the internationalization problem.

Web translating services, such as Google Translate [12] provide an accessible method of automated translation. However, the quality of such translation is not as good as the translation performed by a human [13].

Hunt [14] conducted a survey of email applications aimed at children for multi-lingual support. He reported that only one out of eighteen email applications had an interface specifically designed for children and offered multi-lingual support in few languages. Hunt has developed an email application for children in which parents can provide translation of user interface in a new language [2].

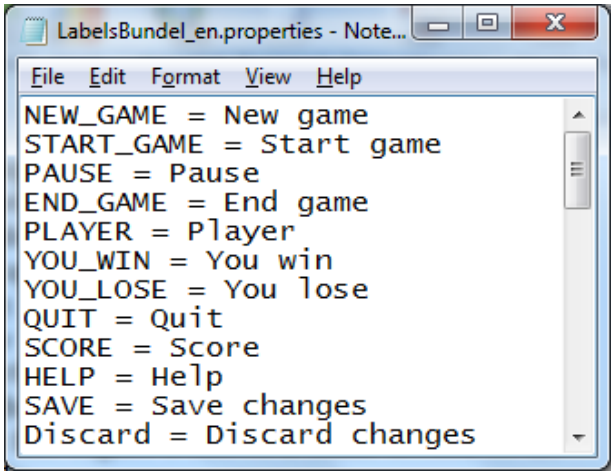

Figure 1: Part of an English property resource file showing the keys and phrases to be retrieved by the system.

III. Design and Implementation

Software producing companies can provide multi-lingual support for only few major languages. This limits the availability of the software products to major communities only. Providing multi-lingual support for minor communities is too expensive for companies and is not scalable. We investigated the idea of providing the ability for end users or their guardians to change a given translation, or create a new translation.

For internationalization and localization, the Java programming language provides the concepts of Locales[15] and Resource Bundles[16]. A locale object is used to represent a particular language and region (or country) while a resource bundle contains words or phrases translated into a particular language. Multi-lingual systems first load a particular resource bundle with the help of a locale object specified by an end user. When the code needs to display a text string to the user, the code finds the appropriate string from the resource bundle by supplying a key. For example, if the code needs to display the English phrase “Start game”, instead of the string “Start game”, being directly displayed, the key ‘START_GAME’ is used to retrieve the string “Start game”, from the English language resource file. Figure 1 shows a section of the English resource file.

When the user chooses a different language, all the text to be displayed will be retrieved from the resource file of that language using the same keys. Figure 2 shows the resource file for Pashto language translation. Pashto is spoken in Afghanistan and in the Khyber Pakhtunkhwa province of Pakistan.

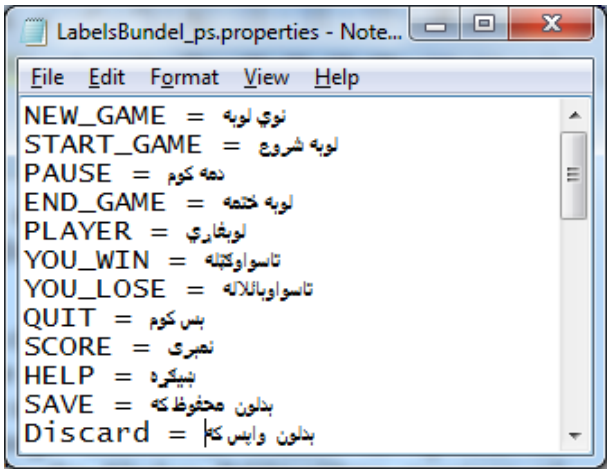

Figure 2: Part of a Pashto property resource file showing the keys and phrases to be retrieved by the system.

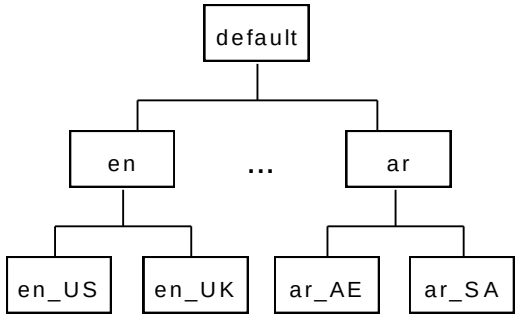

Figure 3: Locales and resource bundles hierarchy.

Proc. of the Intl. Conf. on Advances in Computing and Information Technology-- ACIT 2014

ISBN: 978-981-07-8859-9

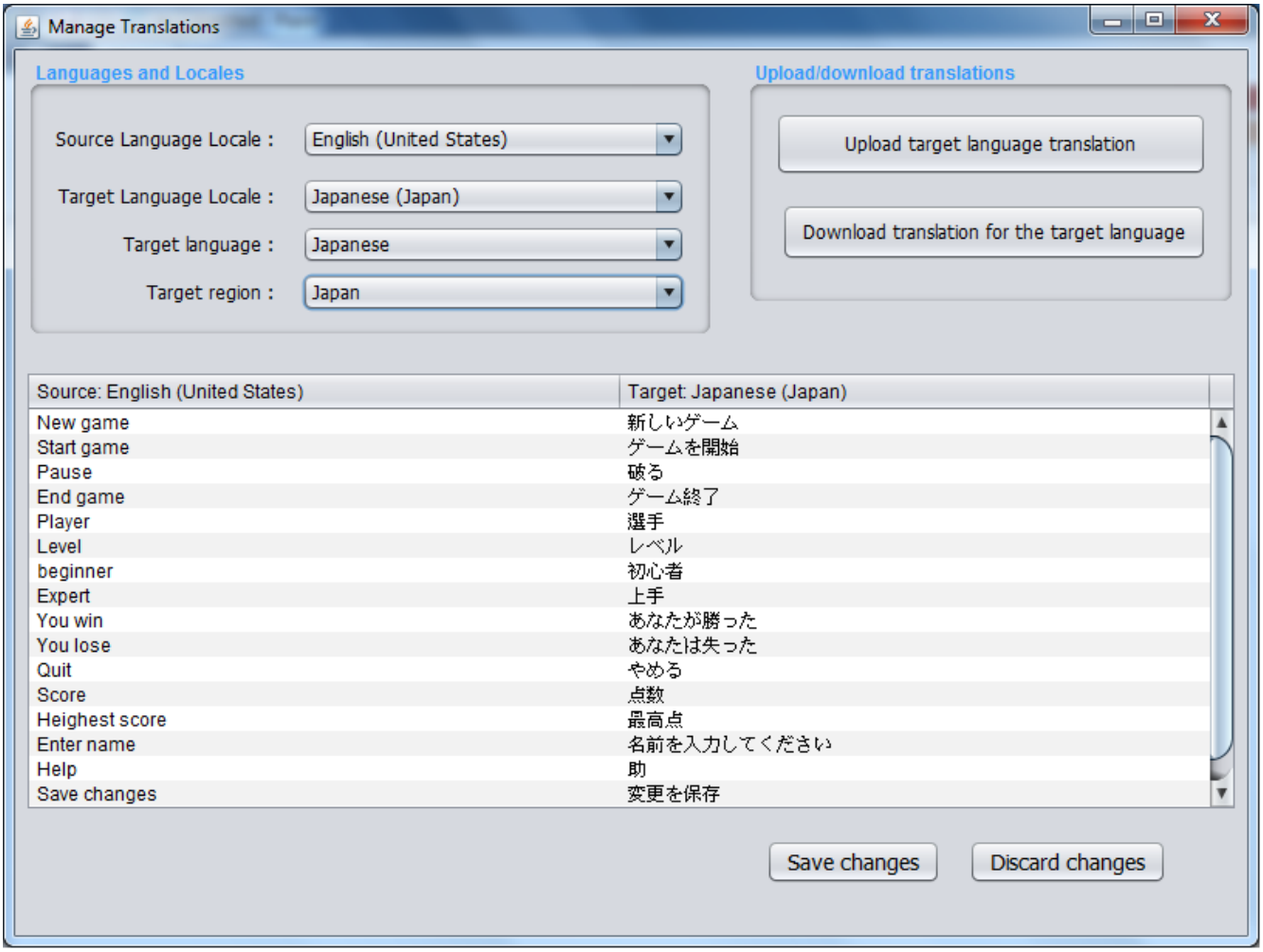

Figure 4: Creating or updating user interface translations

The Java resource bundle architecture provides a robust mechanism for dealing with missing translations. When code attempts to find the string from a resource file for a key that does not exist, it will attempt to obtain the string from another resource file. The resource file chosen will be the one that is of the same base language but with no country specified. If it still fails, it will then revert to the default language file where the key should exist, but the string returned will be in the default language (English) rather than the required language. It works this way because locales and resource bundles are organized in a hierarchical structure, such as the one shown in Figure 3. When an exact match is not found, then the most specific locale is used following the hierarchy from the root (default).

The Java resource bundle architecture requires that resource files for each target language should exist together with the internationalized software product. These resource files are traditionally supplied by software vendors. As indicated earlier, this approach is not scalable if we wish to have support for wide range of languages, especially those spoken by minority groups. Our approach is to allow users of internationalized software to provide translated phrases for all displayable text in the software and save it as a new resource file. Users can then choose this new language for the user interface of the software and can also share it with other users.

IV. Translation Creation

To demonstrate our idea, we have developed a small Java game application for kids. The default user interface is in English language which is provided as a resource bundle. The application allows users to provide translations in the language of their choice for the text phrases used in the application and save it as a new resource file. Figure 4 shows a screen shot of the Graphical User Interface (GUI)

ISBN: 978-981-07-8859-9

that a parent or guardian can use to create a new language resource file for the game application.

In Figure 4, the top-left area (Languages and Locales) allows users to choose the source and target languages by the Locale objects. Source language is one of the languages that is supplied by the vendor along with the software as a resource bundle file. Target language is either an existing language for which the translation is updated, or a non-existing language for which translation is created. In the former case, the target language is selected by using the combo box labeled with 'Target Language Locale'. In the later case, because a Locale object does not exist, the language is selected by using the two combo boxes labeled as 'Target language' and 'Target region'. In the lower area, the left column shows all the phrases in the source language, and the right column shows the same phrases translation in the target language. In the case of new translation, the right column will be blank initially. Users can update the Unicode text entries in the right column and can save their work. The top-right area allows users to share translation files with other users. The idea is that internationalized software should have a corresponding web address that has resource bundle files for different languages. Registered users can upload their saved translation files, or download translation files shared by other users.

V. Conclusions

Internationalized software, which has localized versions for different user communities, has a wider range of audience. The major activity in internationalization is providing multi-lingual support for the user interface part of software. Software producing companies usually provide user interface translations in major languages only as resource bundle files. They cannot afford to provide translations for all languages as it expensive. This results in limited availability of the software to small communities who speak minor languages. Allowing users to create translations of software user interface and sharing it with other users is a viable option. It is scalable as it does not cost much. The only requirement is that the software should allow users to create or import resource bundle files for the languages of their choice.

References

[1] Simultrans. (2013). Localization Return-on-Invetment. Retrieved on Nov 1, 2013, from Simultrans: your languages - yourtimeline: http://www.simultrans.com/education/articles/27-projectmanagement/32-localization-roi

[2] Hunt, T. (2013). Cost effective software internationalisation. Journal of Applied Computing and Information Technology, 17(1).

[3] Howe, J. (2006). The Rise of Crowdsourcing . Retrieved on Nov 1, 2013 from Wired: http://www.wired.com/wired/archive/14.06/crowds.html?pg=1&topic=crowds&topic_set=

[4] Uren, E., Howard, R., & Perinotti, T. (1993). Software Internationalization and Localization: An Introduction. John Wiley & Sons, Inc.

[5] Andrew D., & David C (2001). Java Internationalization. O'Reilly & Associates; ISBN-10: 0596000197, ISBN-13: 978-0596000196

[6] Bert E. (2000). A Practical Guide to Localization. John Benjamins Pub Co; ISBN-10: 1588110060, ISBN-13: 978-1588110060

[7] Liang, Y. D. (2011). Introduction to Java Programming. New Jersey: Pearson Higher Education.

[8] Khaddam, I., & Vanderdonckt, J. (2011). Flippable user interfaces for internationalization. Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems (pp. 223-228). Pisa: ACM.

[9] Peng, W., Yang, X., & Zhu, F. (2009). Automation technique of software internationalization and localization based on lexical analysis. Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human (pp. 970-975). Seoul: ACM.

[10] Wang, X., Zhang, L., Xie, T., Mei, H., & Sun, J. (2009). Locating Need-to-Translate Constant Strings for Software Internationalization. International conference on software engineering (pp. 353-363). Vancouver.

[11] Rößling, G. (2006). Translator: A Package for Internationalization for Java-based Applications and GUIs. ITiCSE'06 (p. 312). Bologna.

[12] Google Translate API. Retrieved on Nov 1, 2013 from Google Developers: https://developers.google.com/translate/

[13] Stefansson, I. (2011). The advancement of Google Translate and how it performs in the online translation of compound and proper nouns from Swedish into English. Retrieved on Nov 1, 2013 from Karlstads Universitet: http://kau.diva-portal.org/smash/get/diva2:440909/FULLTEXT01

[14] Hunt, T. D. (2011). Any language you choose: internationalization of a children's email application. International Conference on Engineering and Information Management, (pp. 34-38). Chengdu.

[15] Java API documentation : Locale class. http://docs.oracle.com/javase/7/docs/api/java/util/Locale.html

[16] Java API documentation : ResourceBundle class. http://docs.oracle.com/javase/7/docs/api/java/util/ResourceBundle.html