2007 International Conference on Convergence Information Technology

Multilingual Programming Language Environments for Intercultural Collaboration of Programming Education in K-12

YongChul Yeum, DaeYoung Kwon, SeungWook Yoo
Department of Computer Science Education, Graduate School
Korea University
Anam-dong Sungbuk-ku, Seoul,136-701, Korea
{yongchul.yeum, daiyoung.kwon, yoosw0810}@inc.korea.ac.kr

WonGyu Lee
Department of Computer Science Education, Colledge of Education
Korea University
Anam-dong Sungbuk-ku, Seoul,136-701, Korea
lee@inc.korea.ac.kr

Susumu Kanemune
Hitotsubashi University Computer Center
Hitotsubashi University
2-1 Naka, Kunitachi City, Tokyo, Japan
kanemune@cc.hit-u.ac.jp

Yasushi Kuno
Graduate School of Business Sciences
Tsukuba University
3-29-1 Otsuka, Bunkyo-ku, Tokyo, 112-0012, Japan
kuno@gssm.otsuka.tsukuba.ac.jp

Abstract

This article is chiefly concerned with multilingual programming language and system for collaboration of programming education in K-12. For a programming language to become a multilingual, it could be programmed in a manner independent on any locale. And this locale independent resource files can be easily translated into a new locale. Therefore when a multilingual programming language for K-12 is designed, it should have no keywords that have a particular meaning to the programming language. In addition, it specifies a set of characters which are based on Unicode character instead of a single character. Dolittle[1] as an both multilingual and educational programming language has all kinds of features descibed above. Especially, it supports translation server for interculture collaboration between Korea and Japan to teach and learn programming education in K-12. We also conducted experimental lesson to evaluate the effectiveness of Dolittle's inflected dictionary in Korea. It was found from the result of the experimental lesson that using inflected dictionary of identifiers can decrease overall error ratio in comparison with using only one identifier.

1 Introduction

This paper discusses multilingual programming environments for programming education and intercultural collaboration in K-12. Our research is also based on a recognition that programming education at K-12 can benefit many diverse people who will live in information society of 21th century. That is, programming experience as a part of IT education allows students to get a better understanding of software and principles of computer science, which is an essential part of computers[9][22].

Since the early 1960's, researchers have built a number of programming languages and environments with the intention of making programming accessible to a larger number of people[11]. Especially we call such a programming language as an educational programming language, i.e. EPL[12][13], such as LOGO, Squeak, Karel. EPL is designed primarily as a learning instrument and not so much as a tool for writing real-world application programs[2].

However, most EPL does not support multilingual programming environment. It means everyone who programs with EPL has to learn English eventually to get a handle on its function names and language constructs. But considering that learning another language is difficult for most people[17] and the language should avoid using words and

0-7695-3038-9/07 $25.00 © 2007 IEEE
DOI 10.1109/ICCIT.2007.292

symbols that are unfamiliar to users[15], it needs to support multilingual environments above all.

Typically, for a programming language to be a multilingual, it should be internationalized and localized through making resource files which are loaded during program execution as needed. This means that the code should be programmed in such a manner that is doesn't dependent on any specific locale - that is language or cultural conventions[18]. Thus to support multilingual languages, one would design programming language to select the relevant language resource files at runtime, which are translated to the required languages. However this kind of methodology has some inherent disadvantages. For instance, if a message displayed to the user in one of several languages is modified, all of the translated versions must be changed[3]. It means the distributed resource files is hard to be modified. Moreover, keywords can't be localized without compiling source code again. Keywords are a word or identifier that has a particular meaning to the programming language or a reserved word which identifies a syntactic form.

In this paper, we suggest keywordless EPL such Dolittle which is our research target. Dolittle[1] has elastic and effective multilingual features. It has no any reserved word by itself. Currently, Dolittle system for Japanese, English and Korean are distributed. Also Dolittle can include user domain-specific dictionary into the user's code or config file. For example, if users are children, teacher can remake the dictionary file according to their knowledge from the domain that are more familiar and appropriate. Moreover, simple word-by-word substitution can transfer a Dolittle program from one language to the other, leading to international program exchange server[9][10].

We also conducted experimental lesson in order to evaluate the effectiveness of Dolittle's inflected dictionary in Korea. These results lead us to the conclusion that when we have a lesson at k-12, student's background knowledge and domain-specific demands[20] need to be handled in detail[7][12][13].

In this paper, we first describe internationalization of programming Language. Secondly we discuss about the keywordless EPL. And then we analyze the features of multilingual EPL, Dolittle. Finally, we introduce experimental lesson about using inflected dictionary of identifiers by Dolittle.

2 Internationalization of Programming Language

Multilingual computing [6] is defined as "Use of computer to communicate with people in various languages". As one of multilingual computing, multilingual programming language makes it possible to teach and learn computer programming among interculcural schools collaboratively. In general, in order to support more than one language simultaneously, it should follow the typical process for making a program multilingual. That is, it needs to take internationalizational process enough to run correctly in any locale. The required corollary to internationalization is localization-the process of arranging for a program to run in a specific locale.

Internationalization is often abbreviated as I18N (or i18n or I18n) where the number 18 refers to the number of letters omitted. Localization is often abbreviated L10n or l10n in the same manner[3].

2.1 Typical Process for I18n

There are several distinct steps to the task of internationalization. Typically we can devide the process for i18n into 3 steps[18][19].

First, source code must be internationalized. This means that we can program the code with an any specific locale. If not internationalized at the beginning of design, supporting multilingual environments could be nearly impossible.

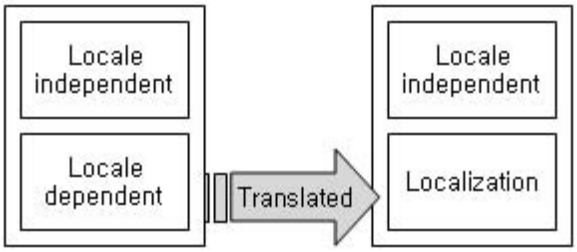


Figure 1. Sample locale independent source and locale dependent source

Second, separate textual data and other environment-dependent resources are separated from the program code shown in Fig. 1. Supporting a different environment, ideally, only requires change in those separate resources without code modification. It means that a language developer deals with the source code and corresponding resource files handled by translator. It seems to be difficult because the development team needs someone who understands foreign languages and cultures; such a person may be difficult to find.

Finally, after localized, we can distribute the localizaton source to the users. This program must be able to read, write, and manipulate localized text, and display all user-visible text in the local language.

2.2 What is Localized?

The hardest work of the localization is often basic translation of text according to the specific locale. This kind of test can be constructs of language(e.g. token, keyword) and messages(e.g. error messages). So what is important is character and character sets which can be used in regular expressions, in the helpers and Tokens sections of the specificaton file.

For example, Java uses the Unicode character encoding, which by itself is a huge step toward internationalization. In addition, the InputStreamReader and OutputStreamWriter classes convert text from a locale-specific encoding to Unicode and from Unicode to a locale-specific encoding, respectively[4].

However, most programming language can not allow reserved words(e.g. if-statement, the words "if" and "else") to be localized. As for EPL, it is very important that everything could be localized without exception. We will discuss the cause why these reserved words(keywords) can not be easily localized in the following chapter

3 Keywordless EPL

So far, there has been little study that tried to make a programming language both keywordless and multilingual for K-12. An example of keywordless progamming language is J programming language[5]. It requires only the basic ASCII character set in order to avoid the problems faced by the special character set of APL. The code of J could be programmed only by ASCII character set. But J language does not support multilingual. Another example is Dolittle[1] which does not has any reserved word(keyword).

Dolittle will be examined further in the next chapter. Now, Let's discuss what is keywordless and what is the merit of being keywordless in detail.

Programming language definition can be loosely divided into two parts:syntax, or structure, and semantics, or meaning[14]. An issue closely related to the syntax of a programming language is its lexical structure-the structure of the words of the language, which are usually called tokens. In the example of a C if-statement, the words "if" and "else" are token. Other tokens in programming languages include identifiers(or names), symbols for operations, such as "+" and "¡", and special punctuation symbols such as the semicolon(";") and the period(".").

Special words in programming languages are used to separate the syntactic entities of programs such as if = 'if' or while = 'while'[8]. These words are classified as reserved words, but in some they are only keywords. Therefore a keyword is a word or identifier that has a particular meaning to the programming language. A reserved word is a special word of a programming language that can not be used as a name[16].

As described above, tokens consist of symbols, identifiers and words which can be as keyword. If keywords, i.e. reserved word, are removed from tokens, tokens are either symbols or identifiers. As symbols are common notation, we only have to deal with identifiers for intercultural and multilingual purpose. Therefore, what is keywordless means that we can deal with program or collaborated with identifier based on Unicode character set.

4 Multilingual EPL Dolittle

Dolittle[1] is an object-oriented and interpreter-based programming language for K-12 schools. Currently, it supports Japanese, Korean and English as shown in Fig. 2. In addition, Dolittle does not have any reserved word or keyword by itself[9]. Only it has predefined names of standard objects. These names can be also extended by adding corresponding alternatives according to the originally representative name [7]. We believe that this keywordless feature of Dolittle can be the key of solving the problems about i18n, which are descrived above.

4.1 Characters and Character Sets

Dolittle lexers generated by SableCC[8][21] can read 16 bit Unicode character streams. A character set in Dolittle's grammar consists of a range of character based the Unicode ordering. It only specify a set of characters, instead of a single character which can be regarded as keyword, i.e. reserved word. Thus Dolittle can include all Unicode characters with an index greater and less, which is an identifier. The following character sets are the Unicode ordering ranges in Dolittle.

- kanji = [0x4e00..0x9fa5];
- hirakata = [0x3040..0x30ff];
- hangul = [0xac00..0xd7a3];
- ascii = [0..0x007f];

A hexadecimal number represents the Unicode character with the same index. So Dolittle can accept any characters among square brackets as identifier. Thus, the most important point for a multilingual programming language is above Unicode based character sets.

4.2 Predefined Names of Standard Objects

As we said above, there is no reserved words in Dolittle. Instead, there are predefined names of standard objects(e.g.

Figure 2. Screens from the Japanese, Korean and English version

Turtle, Timer) or methods (e.g. Forward, Execute) through system dictionary, as shown in Fig. 3. These all predefined names are also composed by Unicode character streams. By changing this dictionary, we can easily convert to other languages.

Dolittle can easily extend its localized word dictionary. Fig. 4. show the process. That is, once after source has been completed, first localization is accomplished by developer according to the base of word dictionary. Then, this first localized files are distributed to the each locale. We call it as base dictionary of locale, which acts as a representative word. In Fig. 4, Circle having locale text means first localized files. This file also can be used on supporting international program exchange server. Circles in a dotted line indicate that they could be translated since translation

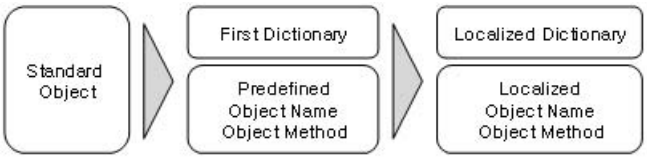


Figure 3. Localization of standard object

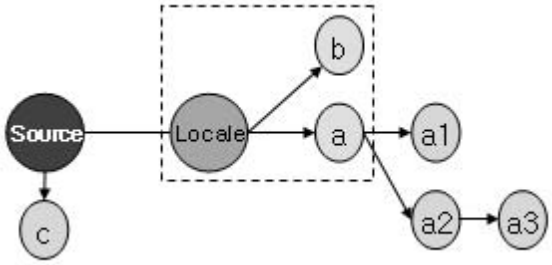


Figure 4. Translation to other languages according to the dictionary

server only know the base dictionary of locale. For example, a circle can be Korean and b circle can be English locale. Futhemore, Fig. 4 shows multi-locale files. That is, if some users are one of other domain in the same locale, i.e. childeren in elementary school, the locale can be reformed as a1 or a2 circle for being more familiar and effective dictionary to them.

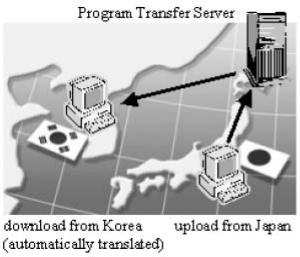
4.3 International Program Exchange

Simple word-by-word substitution can transfer a Dolittle program from one language to the other, leading to international program exchange. User can upload a program from Dolittle and can download with language translation. So we can collaborate and share with each other by using this exchage server. Fig. 5. shows a structure of Program Transfer Server and Translation Table. Program Transfer Server and Dolittle clients communicate with XML-RPC protocol.

Using Program Transfer Server, we can communicate with some students among Korean and Japanease. Fig. 6. shows network button.

5 Experimental Lesson about Using Inflected Dictionary of Identifiers

To evaluate the effectiveness of Dolittle using inflected dictionary of single Korean word, we conducted small experimental lesson with fresh students of Korea University in 2006. The participants in this lesson were ten in all and



English	Japanese	Korean	alias
collision	衝突	충돌	
click	動作	클릭	
root	ルート	루트	
turtle	タートル	거북	거북이
timer	タイマー	타이머	시계
button	ボタン	버튼	
field	フィールド	필드	텍스트필드 빈칸
list	リスト	리스트	목록

Figure 5. Program transfer server and translation table

Figure 6. Screen on network load

learned C language during first term. We divide this students into two groups. Intetionally we made the second group(1:N) know the inflected dictionary of single Korean word, but the first group(1:1) did not. Table 1 shows the curriculum of the lessons and Fig. 4 also shows a screen from the lesson.

Figure 7. Screen from the lesson

Prior to this lesson, we already revised Dolittle[21] version which can automatically make all messages saved into log file when students click the run button. This log file contains all kinds of situation either about errors or not. Table 2 shows the results of analysis of log files.

Table 1. Curriculum of experimental lessons

Lesson	Contents
1	Basic syntax of Dolittle
2	Using predefined objects and methods
3	Method definition and the concept of cloning

Table 2. Analysis of log files

ratio	first group(1:1)	second group(1:N)
all error	44.3%	32.0%
systatic error	48.2%	55.3%
semantic error	21.4%	14.2%

As more than 10% of the students did not make an error as shown in Table 2, it seems reasonable to conclude that using inflected dictionary of identifiers can decrease overall error ratio in comparison with using only one identifier. what is more, decrease ratio in semantic error suggests that it has close relation with user's preference or acquaintance about the words.

6 Conclusions

Producing real multilingual programming language is a complex task. Moreover, the process of making multilingual EPL[12][13] acceptable and adaptable to students of K12 across different nations is a challenging work. Although researchers have built a number of programming languages and environments with the intention of making programming accessible, little attention has been given to the multilingual EPL.

In this paper, however, we have examined the features of Dolittle as a multilingual EPL and resulted the following conclusions.

When a multilingual programming language for K12 is designed, it should have no keywords that have a particular meaning to the programming language. In addition, it specifies a set of characters which are based on Unicode character, instead of a single character.

If a programming language support keywordless aspects and Unicode characters, there are some kinds of advantages as follows. First, localization could be carried out conveniently by locale user. Once resource file localized by developer, locale user can extend first word dictionary according to their need. That is, it can be adapted to user's knowledge and domain-specific request. Second, students in K12 can use translation server such as Dolittle Program Transfer Server for international program exchange and interculture

collaboration about programming education. User can upload program and download with language translation.

Finally, we also conducted experimental lesson in order to evaluate the effectiveness of Dolittle's inflected dictionary in Korea. It was found from the result of the experimental lesson that using inflected dictionary of identifiers can decrease overall error ratio in comparison with using only one identifier.

7 Acknowledgments

We are grateful to Prof. Susumu Kanemune at Hitotsubashi University and Yasushi Kuno at Tsukuba University in Japan for his many comments and contributions during this work. We alse wish to thank to the students of Korea University involved in this experimental work.

This research is supported by Joint Research Project under the Japan-Korea basic scientific cooperation program for FY 2005-2006

References

[1] http://dolittle-eng.eplang.jp/.

[2] http://en.wikipedia.org/wiki/Educational_programming_language.

[3] http://en.wikipedia.org/wiki/Internationalization_and_localization.

[4] http://java.sun.com/.

[5] http://en.wikipedia.org/wiki/J_programming_language.

[6] *An Introduction to Multilingual Computing*. http://languagelab.csumb.edu.

[7] H. S. Choe, D. Y. Kwon, Y. C. Yeum, S. W. Yoo, and W. G. Lee. Multi-reserved words supporting system for object-oriented educational programming language dolittle. The Journal of Korean Association of Computer Education 8(2), 2005.

[8] E. Gagnon. Sablecc : An object oriented compiler framework. Technical report, McGill University. Master Thesis, 1998.

[9] S. Kanemune and Y. Kuno. Dolittle : an object-oriented language for k12 education. Eurologo 2005, Warsaw.

[10] S. Kanemune, T. Nakatani, R. Mitarai, S. fukui, and Y. Kuno. Dolittle : Experiences in teaching programming at k12 schools. The second International Conference on Creating, connecting and Collaborating through Computing, IEEE, 2004.

[11] C. Kelleher and R. Pausch. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys*, 37(2):83–137, June 2005.

[12] H. S. kim, H. Jang, H. C. L. , D. Y. Kwon, Y. C. Yeum, S. W. Yoo, H. C. Kim, and W. G. Lee. Teach programming to non-cs major students : Experiments with storymaking approach. Information Processing Society of Japan, SSS2004, 2004.

[13] D. Y. Kwon, H. M. Gil, Y. C. Yeum, S. W. Yo, S. Kanemune, Y. Kuno, and W. G. Lee. Application and evaluation of object-oriented educational programming language dolittle for computer science education in secondary education. The Journal of Korean Association of Computer Education 7(6), 2004.

[14] K. C. Louden. *Programming Languages: Principles and Practice*, volume 1. Thomson, 2th edition, 2003.

[15] J. F. Pane and B. A. Myers. The influence of the psychology of programming on a language design: Project status report. 12th Workshop of the Psychology of Programming Interest Group, April 2000.

[16] R. W. Sebesta. *concepts of Programming Languages*, volume 1. Pearson, 6th edition, 2004.

[17] D. C. Smith, A. Cypher, and J. Schmucker. Making programming easier for children. October 1996.

[18] M. Suodenjoki. Introduction to internationalization and localization. http://www.suodenjoki.dk/.

[19] O. Tykhomyrov. Introduction to internationalization programming. In *Linux Journal*. http://www.linuxjournal.com/node/6176/print.

[20] W. H. Whang and K. M. Kim. An application of education programming language dolittle in teaching and learning of mathematics. Information Processing Society of Japan, SSS2004, 2004.

[21] Y. C. Yeum, H. S. Jang, D. Y. Kwon, S. W. Yoo, S. Kanemune, and W. G. Lee. Dolittle:a heuristic approach to improving error messaging module based on error feedback strategy for k12. *IJCSNS International Journal of Computer Science and Network Security*, 6(7), 2006.

[22] S. W. Yoo, K. A. Kim, Y. Kim, Y. Yeum, S. Kanemune, and W. Lee. Empirical study of educational programming language for k12: Between dolittle and visual basic. *IJCSNS International Journal of Computer Science and Network Security*, 6(6), June 2006.