Working Paper Series
ISSN 1170-487X

Internationalising a Spreadsheet for Pacific Basin Languages

by Robert Barbour and Alvin Yeo

Working Paper 97/17
July 1997

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

Internationalising a Spreadsheet for Pacific Basin Languages

Robert Barbour
Science, Mathematics and Technology Education Research Centre
University of Waikato, Hamilton, New Zealand
Email: r.barbour@waikato.ac.nz

Alvin Yeo
Department of Computer Science
University of Waikato, Hamilton, New Zealand.
Email: awy@cs.waikato.ac.nz

Abstract

As people trade and engage in commerce, an economically dominant culture tends to migrate language into other recently contacted cultures. Information technology (IT) can accelerate enculturation and promote the expansion of western hegemony in IT. Equally, IT can present a culturally appropriate interface to the user that promotes the preservation of culture and language with very little additional effort. In this paper a spreadsheet is internationalised to accept languages from the Latin-1 character set such as English, Mäori and Bahasa Melayu (Malaysia's national language). A technique that allows a non-programmer to add a new language to the spreadsheet is described. The technique could also be used to internationalise other software at the point of design by following the steps we outline.

Keywords

Internationalisation, spreadsheet, cultures, Pacific languages, human computer interaction, western hegemony, culturally appropriate.

1. Introduction

In this paper, we explore techniques for providing information technology (IT) for markets that are small or culturally-distant from Europe and North America. As the nations of the world join in trade and commerce, smaller and culturally-distant nations are at risk of being engulfed in the head start acquired by the developed world. We have selected the electronic spreadsheet as the focus of our research because it is the computing tool of choice for business and commerce. In the search for techniques to promote the adoption of information technology, we are aware that without culturally appropriate software, the languages and cultures of minority groups may not survive the adoption of IT. We hold this view because modern societies support trade and commerce by using information technology in the form of the spreadsheet for managing monetary transactions.

In the first section, we describe the dual processes of internationlisation and localisation in general terms. Techniques for modifying a simple MS DOS spreadsheet are then described in section two. The limitations and results of these techniques are identified to exemplify current strategies for internationalising software in sections three and four. Further work is suggested prior to a conclusion which sketches some implications of the new techniques.

1.1 *Internationalisation and Localisation*

Software *internationalisation* refers to the "process of making a system or application software independent of, or transparent to, natural language. If a system can support any language, then it is fully internationalised; if it supports only a limited subset of languages, then it is partially internationalised" (Unicode,1995). Once an application has been internationalised, it may then be localised to meet the needs of particular cultures.

1.2 *Why internationalise for the Pacific Basin?*

The Pacific Basin is a vast ocean-dominated saucer-shaped area dotted with one or two larger islands and thousands of smaller ones. This huge area has a low overall population density but a great number and diversity of languages and cultures. The richness and variety of cultures represents a pool of cultural experiments from which the rest of humanity may wish to draw at some time in the future. Thus, there are purely selfish human reasons for promoting a range of culturally appropriate software solutions that contribute to ensuring the survival of the linguistic and cultural variety in the Pacific Basin. Without the medium of language, distinctive cultures will disappear. Griffiths, Heppell, Millwood and Mladenova (1994) believe that by creating software for the minority communities, we are preserving the cultures and indirectly preserving potential solutions to future problems. For example, if China did not create software which uses Hanyi Pinyin, all the innovative input methods and output mechanisms (Huang,1985; Becker, 1985; Walters,1990; Hsu,1991) may not have materialised. These input methods are required to key in the 50,000 characters in the Chinese script using the American keyboard. These input methods can be categorised into input by: whole characters, phonetic values of the characters, the arrangement of the radicals within a character and the partitioning of characters into radicals or similar patterns (Huang, 1985). In exploring the issues of providing IT across cultures we are aware that the 'traffic' can be both ways, that is majority and minority cultures can contribute to the growth of IT solutions. While current solutions, such as those discussed below, depend upon benevolence or the commercial necessities faced by western companies, we would argue that the interaction can be mutually rewarding if mechanisms are provided that

enable the two-way flow of IT. Just as monetary transactions and trade advantage both partners we argue that agreed mediums of exchange for IT should be developed.

There are, not unsurprisingly, clear economic rewards for internationalising software. Savings are identified by O'Donnell (1994) as: fewer code revisions, less delay in reaching users, reduced operational inconsistencies, simplified extension model, simplified program management, cost savings potential for expanded markets and simplified maintenance.

Most software companies internationalise their software to reach a bigger market and ultimately to make more profit. For example, Microsoft Windows supports most spoken languages in the major cultures of the world (Kano,1995). The Apple Macintosh operating system, which covers more than 30 different localised versions, also supports most majority languages (Apple,1992). However, languages of minority groups such as Mäori, Bahasa Melayu (Malaysia's national language), Fijian and Samoan have been neglected. These languages have been omitted as the markets in which they are located are not large enough to be economically feasible and profitable to the software companies.

In countries where English is widely spoken in association with indigenous languages, there is a high likelihood that the software in English will be made available and used because the language of commerce is English. For example, in Malaysia, computer classes might be taught in Bahasa Melayu and use a reference book in Bahasa Melayu but the software used is most likely to have an English user interface. In New Zealand, little software is available in Mäori, partly because there is no institutional support and partly because of the hegemony of North American software companies over the New Zealand software market. On the other hand, in countries where English is not as widely used, software has been created for local consumption. For instance, in China, Thailand, Korea and Japan, research and development resulted in computer systems and software designed for those languages and cultures. Thus, minority languages in countries where English is widely used are at risk of being left off the technology bandwagon. Although steps have been taken in many countries such as Malaysia to ensure that language keeps up with technology by translating computer terms into the minority language, these steps will be unsuccessful if no language specific software is being created. Furthermore, as suggested above, we see the move to promote the development of minority software as essential for the growth of the IT industry. As the control over the industry exercised by IT giants such as Microsoft and IBM grows, the variety of solutions generated by those companies is reduced. An industry which is dominated by single solutions is vulnerable to malicious activities.

1.3 *Our reasons for internationalising*

We decided to research the software internationalisation process to familiarise ourselves with the internationalisation problems and techniques for solutions of these problems. Only by internationalising a particular software package can we appreciate the complexity of the process. In addition, we wanted to create a tool that could be used by a number of cultures in the increasingly multicultural context of New Zealand schools. The cultures targeted are those which have been identified as being at risk of being left behind by technology. Furthermore, the internationalised software was designed as a research tool to support multi-cultural or cross-cultural information technology research. Features of the software chosen for this research are detailed in the Methods section to follow.

1.4 *Approaches to internationalisation*

Taylor (1992) describes three approaches to internationalisation:

compile time internationalisation
link-time internationalisation and
run-time internationalisation

We will use a simple program to illustrate the differences between these types of software internationalisation. The example program has a module which will hold the messages such as "Hello Jonah" (see below) and a module for printing messages to the screen. In addition to the program that prints "Hello Jonah", we would like a Mäori and a Bahasa Melayu version.

```
Program printMsg;
Begin
        print("Hello Jonah");
End;
```

Compile-time internationalisation simply means changing the text in the source code. To obtain a Mäori version, we change from "Hello Jonah" in the source code to "Tënä koe Jonah". The source code is then recompiled, linked and run as an executable in the conventional way. Likewise, to obtain a Bahasa Melayu version, we alter "Hello Jonah" to "Apa khabar Jonah?", compile and run the executable. Note that three executables are created, one executable for English, Mäori and Bahasa Melayu.

Link-time internationalisation means that the program is provided with a copy of the object file of the printMsg program in Mäori (maaori.obj), in English (english.obj) and in Bahasa Melayu (bmelayu.obj). Thus, to obtain the messages in different languages, the corresponding object file is linked with the object file of the printing module, print_mod.obj. To obtain the English

version, english.obj is linked with print_mod.obj, and then run as the final executable. Similarly, to obtain the Mäori version, the maaori.obj is linked with the print_mod.obj, and then run as an executable. Note again, we end up with three executables.

There are two possibilities for software that has been designed for run-time internationalisation. In Type I, according to Taylor (1992), the language set is provided by the supplier or distributor, whereas in Type II the language is added, in a localisation process, by the enduser. In Type I, a message such as "Hello Jonah" is placed into a text file, language.msg. When a user has the program executed on a machine, a menu appears in which the available languages are displayed. On the selection of a particular language, such as Mäori, the program reads the corresponding language message from the respective language files. For example, "Hello Jonah" is read from the english.msg file, "Tënä koe Jonah" is read from the file maaori.msg, and "Apa khabar Jonah?" from the bmelayu.msg. These message files are external to the source code. One of these message files is selected at run-time. For larger programs, the language selection step initiates a process that subsequently displays accurate strings for all other on-screen messages, from the message file, in that selected language.

The second type of run-time internationalisation is more flexible and convenient for localising software by the enduser. In this approach, all user-interface-language-specific on-screen messages are identified by tokens in the source code. A subprocess reads the message tokens and relates these to other files which contain numeric indices relating string sets to the message tokens. The files containing strings are editable ASCII code for common Latin-1 languages. This approach allows the user to select messages in one language from those available at run-time as described in the following sections on method. Localising an application to a new locale is carried out by adding the strings for the words of the new language. Original text is substituted with the new string set using the technique discussed in detail below.

In types I and II "run-time internationalisation" applications, only one executable is created. Instead of three executables, three text files are used to contain the messages of the different languages. The main difference between Type I and Type II "run-time internationalisation" applications is that in Type II, the enduser can add another language to the software, whereas in Type I, the enduser is limited to the languages provided by the supplier. Note also that we can make corrections to the message files at run-time in the run-time internationalisation executable without having to recompile and relink the program.

In the compile-time internationalisation, corrections can only be made in the source code, which needs to be compiled, linked and then run. In the link-time internationalisation, corrections would need to be made in the object files (in which case, the source code would need to be corrected, compiled, linked and then run).

Currently, Microsoft Excel is available in many (separate) languages such as French, German, Dutch and Swiss. From our understanding of Kano (1995), Microsoft uses compile-time internationalisation. We believe Lotus 123 also uses compile-time internationalisation because the language is determined at installation time and is dependent on the Windows operating system.

A comprehensive list of factors that need to be addressed in the internationalisation and localisation process can be found in articles and books on internationalisation and localisation such as Apple(1992); Russo and Boor(1993); Uren, Howard and Perinotti, (1993); O'Donnell, (1994); Chris Miller, (1994); Kano, (1995); Belge, (1995). Accommodating all these aspects presents a daunting task. Current spreadsheet vendors such as Microsoft Excel and Lotus 123 avoid these issues by exploiting internationalised versions of the Macintosh and Windows operating systems rather than designing custom interfaces. We focussed on the language aspects of internationalising FIRST as detailed below in the Methods section

2. Method

This section describes the goals that were established for the research. Software, which meets the specifications derived from the goals, is described next. How the original source code for the research project was modified, and the steps taken to achieve the internationalisation of the spreadsheet are detailed. During exploratory trials, the spreadsheet (Te Ripanga) was localised to accept Mäori user interface messages using the compile–time technique. The same source code was later internationalised to accept English and Bahasa Melayu simultaneously with the Mäori language. Run-time versions are currently under test with users.

2.1 *FIRST Goals*

With the goal of an internationalised version of a spreadsheet, we used the following specifications to develop FIRST (First Internationalised Research Spreadsheet Tool). FIRST should be able to accommodate a number of Pacific Basin languages simultaneously, for instance, English, Mäori, Samoan, Fijian and Tongan. A keystroke combination is used to select the various languages, e.g. ALT-E for English, ALT-M for Mäori, ALT-S for Samoan, ALT-F for Fijian and ALT-T for Tongan. Changing the selected language was required while the software was running. A further goal was to provide end-user facilities that would allow the addition of messages in other languages (whose alphabets are based on the Latin 1 character set). Thus, a user with no programming skills should be able to add a new language (localise the software) because no modifications are required to the source code of FIRST during the localisation process.

2.2 *Resources*

The spreadsheet used for the research is called TCALC. The source code library of TCALC was bundled as an example program with Borland's Turbo C++ compiler. Permission was sought and obtained from Borland to use and modify their code. TCALC, which has a Lotus 1-2-3-type interface, was chosen as it was a compact spreadsheet that had most of the basic features and functionalities of commercial spreadsheet programs. Like any other spreadsheet, TCALC accepted text, values and formulas. TCALC's functionalities included load, save, and print. TCALC saves the spreadsheet into its own format which is not compatible with other spreadsheets such as Excel or Lotus 1-2-3. However, the print functionality allows the user to print the existing spreadsheet to a text file. Other features include insert rows and columns, delete rows and columns, formula display, automatic recalculation, and use of mathematical functions such as sin(), cos() and tan(). The sum-of-a-range-of-cells function is also provided. TCALC was selected because of the availability of the source code, which meant that we could modify the spreadsheet to suit our research needs. The spreadsheet was modified in various versions using Borland Turbo C++ (DOS Version 3.0) on a Pentium machine running Windows NT and Windows 95. The ensuing sections will describe the steps that were taken to develop FIRST, the internationalised version of TCALC.

2.3 *Compile Time Internationalisation*

A Mäori version of the spreadsheet TCALC, known as Te Ripanga, was used to explore compile-time localisation (Barbour, 1996). That process required that any occurrence of the language sensitive text in the source code user interface was translated to Mäori. A person literate in the discipline of computing and also the Mäori language provided the translation. With this form of internationalisation, Te Ripanga could only use one language. To alter the language, the user interface strings in the source code would have to be modified for a different language and the localised source code recompiled. Several important lessons, learned during this process, are described in the next five sections.

2.4 *Extract All Culturally Sensitive Components*

The objective of this stage was to identify the components in the source code that were either language specific or likely to be culturally sensitive. The language specific components are part of the user interface. Examples of these items include: the menu names, the menu item names, status bar displays of cell contents (text, value, formula), error messages, instructions, commands and dialogue messages. The currency symbol was also identified as culturally specific because it reflected the currency of target cultures each of which used different symbols.

In TCALC, almost all the language-specific components were defined as constants and placed in a header file. Extracting these components was straightforward. However, the more tedious task was to discover the other components that were "hidden" in the source code. These

components included the user-computer interactions such as the selection of Yes or No in response to a requester. Bahasa Melayu and English are used in the following discussion which illustrates the multiple languages used in FIRST. Bahasa Melayu was selected as one of the authors was fluent in Bahasa Melayu and English. Once all the language-sensitive components were isolated, the message tokens (for language sensitive components) of a particular language were placed in a text file. This type of text file is also referred to as a resource file in some literature (Kano (1995); Pugh and LaLonde,(1992)). Appendix A provides examples of message files comprising English, Bahasa Melayu and Mäori message tokens.

2.5 *Translate the culturally sensitive components*

After the source code was modified, the original message tokens (in English) were translated to Bahasa Melayu. Transliteration of the messages is strongly discouraged as the meaning conveyed may be altered. The translation was carried out by two computer literate speakers of Bahasa Melayu with reference to the Istilah Komputer (Dewan Bahasa dan Pustaka, 1993) (a glossary of computer terms in Bahasa Melayu) and a Lotus 1-2-3 reference tutorial (Yong, 1995). Computer literate translators proficient in the target language were used because translation of computing concepts may also be required. Furthermore, a translated command may be grammatically correct but may contain offensive nuances or connotations, which can only be picked up by first language speakers of the target language. During the translation process, the Bahasa Melayu translators collaborated with other speakers who have had spreadsheet training in Bahasa Melayu. This collaboration process was carried out as a precaution in case the principal translators were "led astray" by their contact with English speakers.

In the translation of the message tokens to Bahasa Melayu, the problem of non-existing words did not arise because reference was made to the Istilah Komputer text. Almost all the English computer terms or words used had an equivalent word in Bahasa Melayu. If the word was not found in the Istilah, the Lotus 1-2-3 tutorial book had some alternatives. Slang, humour and jargon were omitted in the translation. Translation of the text from English to another language usually increased the number of characters in the message string. This increase was exemplified in the following instance. There were 974 characters in the English message tokens. The Bahasa Melayu translation had 1169 characters, an increase of about 20%. However, it must be noted that, in some instances, translation may shorten the length of the message string as for example *formula* (English) to *rumus* (Bahasa Melayu). The increase in text length caused changes to the text display on the screen in FIRST. Consequently, the fit of the text on the screen had to be checked. This step is described in Section 2.7 below.

TCALC users select commands from screen menus by pressing the uppercase letters (shortcut keys) of the commands. For example, the list of commands in English is "Spreadsheet, Format, Delete, Goto, Col, Row, Edit, Utility, Auto, Quit". To choose the Delete command, the user presses the D key. After translation however, these shortcuts will be altered. The Bahasa Melayu list of commands became "hamparaN, Format, Hapus, Goto, Lajur, Baris, Sunting, Utiliti, Auto, Keluar". A separate and different list of shortcut keys had to be determined for the Bahasa Melayu version of TCALC. Notice not all the first letters of the commands are used. The N in *hamparaN* (spreadsheet) is used, as opposed to H, to avoid a clash with *Hapus* (delete). The shortcut key H was selected to ensure consistency as *Hapus* (delete) also appears in the delete row and delete column commands.

Once the translation of the message tokens has been carried out, TCALC had to be altered to accommodate other languages besides English. The following section details the steps taken to internationalise TCALC.

2.6 *Modify the software to accept and add other languages*

The next step in the internationalisation process is to modify the spreadsheet so that it can be used with different languages. In the original TCALC, all the message tokens were strings or characters, and declared as constants. In FIRST (the internationalised version of TCALC), all these strings were replaced as variables. By doing so, these variables would contain the messages (of different languages) that were loaded. For example, the variable msgOverWrite will contain the string "The file exists. Do you want to overwrite it?" when the English version is used, and the variable will contain the string "Fail tersebut wujud. Mahukah anda menulisgantikan fail lama?" in the Bahasa Melayu version. Functions such as LoadMsgDb(language) were added to FIRST. Depending on which language was selected, the LoadMsgDb(language) function will read all the message tokens of a particular language into the corresponding message variables. These message tokens can be seen in Appendix A.

Given that FIRST can accommodate a number of languages at any one time, code was added to prompt the user to select a language from a list of languages displayed on the screen at the start of the FIRST session. Once the user had selected the language, other languages can be selected using the ALT-key combination. Figure 1 below shows the LangList.txt which lists the message token files to be used and the designated ALT key to be used. In this instance, pressing ALT-B at run-time will load the Bahasa Melayu version, ALT-E the English version, and ALT-M the Mäori version. When ALT-B is pressed, the message tokens in the message file "bmelayu.msg" are loaded into the message token variables in FIRST. Appendix B shows the different versions of FIRST at run-time.

```
1     "B"   "bmelayu.msg"
2     "E"   "english.msg"
3     "M"   "maaori.msg"
```

Figure 1: LangList.txt containing three message files

Bahasa Melayu: Hamparan
English: Spreadsheet
Mäori: Te Ripanga

Tekan B untuk Bahasa Melayu
Press E to select English
Patotöhia M kia Mäori

Figure 2: Prompt screen with three languages

The number of entries in the file LangList.txt determines the languages that are available. The LangList.txt in Figure 1 will determine the prompt screen in Figure 2, likewise, LangList.txt in Figure 3 will determine the prompt screen in Figure 4. The prompt screen messages are message token 1 and 2 of the message files as dictated by the LangList.txt

```
1     "E"   "english.msg"
2     "M"   "maaori.msg"
```

Figure 3: LangList.txt containing two message files

English: Spreadsheet
Mäori: Te Ripanga

Press E to select English
Patotöhia M kia Mäori

Figure 4: Prompt screen with two languages

2.7 *Ensure text fits the screen*

Microsoft SDK (in Chris Miller,1994) predicts that translation of messages from English into another language will increase the length of the message. In FIRST, the messages did increase the length of the strings as described in section 2.5 above. The main concern related to the length increase is the effect on the screen, that is, some of the longer messages may not fit onto the screen. Thus, all the message tokens in each language were tested during run-time to ensure they could fit onto the available screen real estate. All the Bahasa Melayu translation fitted the screen except for *PerhitunganAuto* which wrapped down to the next line. -The message *PerhitunganAuto* is stored as msgAutoCalc (see message 12 in Appendix A) and is used to indicate whether -automatic recalculation is on. This message token is displayed at the top right of FIRST (see Appendix B for a screen dump of FIRST). When the Mäori version was used, the same problem arose, that is *KaMahi Tätai* (msgAutoCalc) wrapped to the next line. This wrap problem occurred as the message token msgAutoCalc occupied exactly eight spaces from the edge of the screen. As the length of the Mäori and Bahasa Melayu translations was greater than the English word, the words wrapped to the next line. We solved this

problem by displaying the msgAutoCalc (justified to the right) only if there is enough space left after displaying the other message tokens such as msgMemory and msgCommand. If there is not enough space left, the msgAutoCalc is written after the other message tokens. If the word wrapped problem still exists, another alternative would be to shorten or abbreviate the message token involved. The abbreviation should contain the meaning intended in the original message token. Again, the translators need to ensure that the abbreviation is acceptable in the target community and would not offend anyone. The message tokens of all the languages in FIRST were tested to ensure the tokens fitted the available screen real estate.

2.8 *Test with target groups*

After ensuring that all the message tokens of all the languages could fit the screen, FIRST was ready for testing with users of the target community. At present we are carrying out the user evaluation of FIRST. The users may detect any problems missed by the developer. By testing FIRST, we can ensure a higher chance of acceptability by target cultures. Outcomes of enduser testing will be reported in a subsequent paper.

3. Limitations

In FIRST, we can add as many languages as we want, the only limitation being the number of languages that can be displayed on the prompt screen. There exists a more rigid restriction on the number of message tokens we can have also exists. Memory has to be allocated for each of the message token variables. The number of message tokens we can have will depend on the amount of memory available, given that memory is also required for data entered into the spreadsheet.

FIRST can accept languages with alphabets that are Roman characters from the Latin-1 specification. However, languages that are pictographic or script-like such as Chinese, Japanese, Korean, Arabic or Hebrew require a total rewrite of FIRST. The spreadsheet would have to run in a graphics mode instead of a text mode, which it currently does. New input techniques will need to be coded to allow the input of more complicated characters. Techniques for bi-directionality (Arabic and Hebrew) will also be required. It would seem much more economical to design and develop the global internationalised spreadsheet from scratch. That work falls outside the scope of the current research.

4. Results

The internationalised spreadsheet FIRST can be localised by a user without a programming background. Trials indicate that the modified spreadsheet compiled and ran as expected. The new functionality enables another language to be added by translation of the message token

files, and the addition of the relevant ALT-key combination and the message file name into the LangList.txt file. The new functionality means that communities can make use of these language-adding features without having to modify the source code of the spreadsheet. Given that the software will then be available in the native tongue of the enduser, we expect users will learn faster as observed by Griffiths (1994). With the different languages available simultaneously, FIRST could be used to teach basic spreadsheet concepts in a number of languages in multilingual contexts.

5. Further work

A number of minor technical details need further work to enhance the usefulness of the software. Commas are used to separate thousands in numbers in the UK and US. However, in Europe, the thousand separators is a full stop or a space. Similarly the full-stop is used as a decimal indicator in US and UK, but in Europe, the comma is used instead. Thus, the comma may need to be extracted as a message token and the various conventions stored in a message file.

Although DOS uses code page 437, which includes characters with diacritics (e.g. ä, é), these characters can be displayed on screen but not entered via the keyboard in FIRST. In Mäori, the ä can be substituted with aa. Another alternative would be to modify the input module of FIRST such that diacritics can be inputted. With the use of diacritical, FIRST could use French, German and most European languages using the DOS code page 437.
The colours in FIRST could also be extracted as message tokens. The colours can then be altered to suit the target community. For example, some colours are considered bad luck in certain countries and thus could be omitted from the user interface. We can also extract the mathematical functions such as cos(), sin() and cos () as message tokens.
A further project could extend to creating a tokenised programming language for particular subsets of communication domains. Given that the content and concepts are shared in a number of target cultures, efficient software development may require the specification and implementation of tools for managing tokenised programming functionality. Interface components would then be made available Lego™-like on a user definable basis following the model of FIRST.

6. Conclusion

We have suggested that the current trend in IT towards increased western hegemony leaves the industry vulnerable. We have argued that promoting the acceptance of diversity in IT solutions and generating mechanisms for generic solutions will increase the adoption of IT. We believe process of empowering minority groups through placing the responsibility for creating interface details in the hands of the enduser should bring about better user-directed learning

about Information Technology. Against a background of an inevitable increase in the business use of information technology within and between cultures we have shown that the task of internationalising software in a narrow domain of character sets is easily accomplished. Furthermore, FIRST also allows new Latin-1 based languages to be added without having to modify the source code. Given that the modified files are in extended ASCII text, all a user needs to do is type the translation text into the FIRST without the need or use of other programs or utilities. With this flexibility, FIRST may be the first spreadsheet in which minority languages can be seen used. For example, Iban, the language of an indigenous community in Sarawak may now be able to record financial transactions created in the mother tongue. We have also drawn attention to the difficulty of internationalising software intended for markets in which the language does not have a Latin-1 alphabet. The next human computer interface programming challenge will be to find programming mechanisms that integrate ideographic and Latin-1 languages in the same spreadsheet user interface.

Acknowledgments

The authors are indebted to Eunice Yeo, for the Bahasa Melayu translation, and Te Taka Keegan, for the Mäori translation of the message tokens. The authors would also like to thank the referees of the New Zealand Journal of Computing for their critique on an earlier version of this paper.

Biographical Data

Robert Barbour

Bob Barbour has a background in Social Sciences, Education and Computer Science. He is interested in researching software tools which preserve human rights while promoting international communication.

Alvin Yeo

Alvin Yeo is a DPhil computer science student at the University of Waikato. His research interests include software internationalisation and localisation, and culture's effects on human computer interaction. Alvin is also a tutor currently on study-leave from the Faculty of Information Technology, Universiti Malaysia Sarawak.

References

Apple (1992) : *Guide to Macintosh Software Localization*, Addison Wesley, Reading, Mass.

Barbour, R.H. (1996): *Te Ripanga: the Spreadsheet.* Proceedings of the New Zealand Computers in Education Society 6th Biennial Conference Hamilton, New Zealand, pp66-76.

New Zealand Becker, J. D. (1985) : Typing Chinese, Japanese and Korean, *IEEE,* January 1985, pp27-34.

Belge, M. (1995) : The Next Step in Software Internationalization, *interactions,* **2**(1), pp21-25.

Chris Miller, L. (1994) : Transborders Tips and Traps, *BYTE*, **19**(6), pp93-102.

Dewan Bahasa dan Pustaka (1993) : *Istilah Komputer.* Dewan Bahasa dan Pustaka, Kuala Lumpur, Malaysia.

Griffiths, D., Heppell, S., Millwood, R. and Mladenova, G. (1994) : Translating Software: What It Means and What It Costs for Small Cultures and Large Cultures, *Computers in Education*, **22**(1/2), pp9-17.

Hsu, S. C. (1991) : *A Flexible Chinese Character Input Scheme,* UIST '91, (Hilton Head, South Carolina 11-13 November), pp195-200.

Huang, J. K. (1985) : The Input and Output of Chinese and Japanese Characters, *IEEE,* January 1985, pp18-24.

Kano, N. (1995) : *Developing International Software for Windows 95 and Windows NT,* Microsoft Press, Redmond, Washington.

O'Donnell, S. (1994) : *Programming for the Whole World: A Guide to Internationalization,* Prentice Hall, Englewood Cliffs, New Jersey.

Pugh, J. and LaLonde, W. (1992) : Internationalizing your application, *Journal of Object-oriented Programming,* **4**(8), pp59-62.

Russo, P. and Boor, S. (1993) : *How Fluent is your Interface? Designing for International Users,* INTERCHI '93 Conference on Human Factors in Computing Systems: INTERACT '93 and CHI'93, (Amsterdam, 24-29 April), ACM Press. pp342-347.

Taylor, D. (1992) : *Global Software: Developing Applications for the International Market,* Springer-Verlag, New York.

Unicode (1995) : *Unicode and Internationalisation Glossary.* http://www.stonehand.com/unicode/glossary.html.

Uren, E., Howard, R. and Preinotti, T. (1993) : *Software Internationalization and Localization: An Introduction,* Van Nostrand Reinhold, New York.

Walters, R. F. (1990) : Design of a Bitmapped Multilingual Workstation, *IEEE*, Feb 1990. pp33-41.

Yong, S. H. (1995) : *Siri Literasi Komputer Fajar Bakti: Lotus 1-2-3*, Penerbit Fajar Bakti Sdn. Bhd., Shah Alam, Malaysia.

APPENDIX A

Message Tokens in English

```
1       "English: Spreadsheet"
2       "Press E to select English"
3       "Can't open the file"
4       "Press / for the list of commands"
5       "Memory Available:"
6       "ERROR"
7       "Not enough memory to allocate cell"
8       "Empty"
9       "Text"
10      "Value"
11      "Formula"
12      "AutoCalc"
13      "Formula"
14      "Enter the file name of the spreadsheet:"
15      "Press any key to continue"
16      "Enter the new column width:"
17      "The file exists.  Do you want to overwrite it?"
18      "Not enough memory for entire spreadsheet"
19      "That is not a FIRST spreadsheet"
20      "The file does not exist"
21      "Enter the cell to go to:"
22      "You must enter a number from %d to %d."
23      "That is not a legal cell"
24      "Enter the first cell to format:"
25      "Enter the last cell to format:"
26      "The row or the column must be the same"
27      "Do you want the cell right-justified?"
28      "Do you want numbers in a dollar format?"
29      "Do you want commas in numbers?"
30      "How many decimal places should the number be rounded to?"
31      "Do you want to print in 132 columns?"
32      "Enter the file name to print to, or press ENTER to print on the printer"
33      "Print the border?"
34      "Loading..."
35      "Saving..."
36      "Save current spreadsheet?"
37      "Parser stack overflow."
38      "Spreadsheet, Format, Delete, Goto, Col, Row, Edit, Utility, Auto, Quit"
39      "SFDGCREUAQ"
40      "Load, Save, Print, Clear"
41      "LSPC"
42      "Insert, Delete, Width"
43      "IDW"
44      "Insert, Delete"
45      "ID"
46      "Recalc, Formula display"
47      "RF"
48      "YN"
49      "$"
```

Message Tokens in Bahasa Melayu

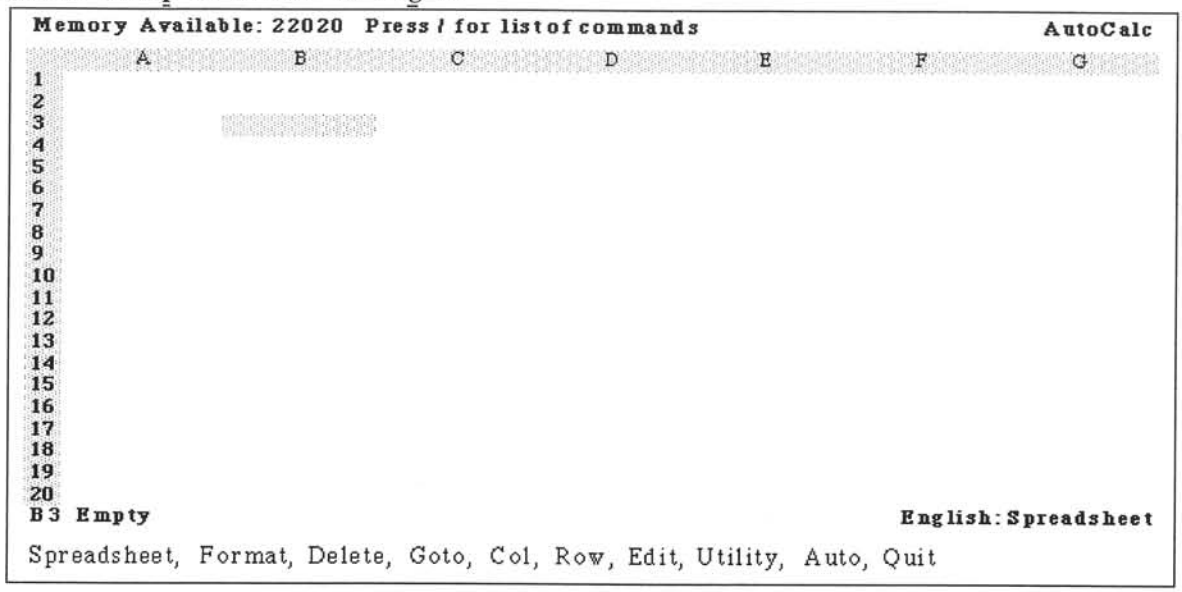
1 "Bahasa Melayu: Hamparan"
2 "Tekan B untuk Bahasa Melayu"
3 "Fail tidak dapat dibuka"
4 "Tekan / untuk senarai perintah"
5 "Memori yang sedia ada"
6 "Ralat"
7 "Tidak cukup memori untuk memperuntukkan sel"
8 "Kosong"
9 "Teks"
10 "Nilai"
11 "Rumus"
12 "PerhitunganAuto"
13 "Rumus"
14 "Taipkan nama fail untuk dimuatkan"
15 "Tekan mana-mana kekunci untuk menjalankan hamparan"
16 "Taipkan lebar lajur baru:"
17 "Fail tersebut wujud. Mahukah anda menulisgantikan fail lama?"
18 "Memori tidak mencukupi untuk seluruh hamparan"
19 "Itu bukan fail hamparan FIRST"
20 "Fail tersebut tidak wujud"
21 "Masukkan sel yang diingini:"
22 "Anda mesti masukkan nombor dari %d hingga %d"
23 "Sel tersebut tidak sah"
24 "Masukkan sel pertama untuk diformat:"
25 "Masukkan sel terakhir untuk diformat:"
26 "Baris atau lajur mestilah yang sama"
27 "Adakah anda inginkan semua sel diselaraskan ke kanan?"
28 "Adakah anda ingin semua nombor dalam format RM?"
29 "Adakah anda ingin semua nombor berkomma?"
30 "Berapa tempat perpuluhan yang perlu dibundarkan untuk setiap nombor?"
31 "Mahukah anda mencetak 132 lajur?"
32 "Masukkan nama fail untuk dicetak, atau tekankan ENTER untuk mencetak."
33 "Cetakkan border?"
34 "Dalam proses memuatkan fail ..."
35 "Dalam proses menyimpan ..."
36 "Simpan hamparan ini?"
37 "Tindanan penghurai melimpah atas"
38 "hamparaN, Format, Hapus, Goto, Lajur, Baris, Sunting, Utiliti, Auto, Keluar"
39 "NFHGLBSUAK"
40 "Muatkan, Simpankan, Cetakkan, Hapuskan"
41 "MSCH"
42 "Sisipkan, Hapuskan, Lebar"
43 "SHL"
44 "Sisipkan, Hapuskan"
45 "SH"
46 "Hitung semula, Tunjukkan rumus"
47 "HT"
48 "YT"
49 "RM"

Message Tokens in Mäori

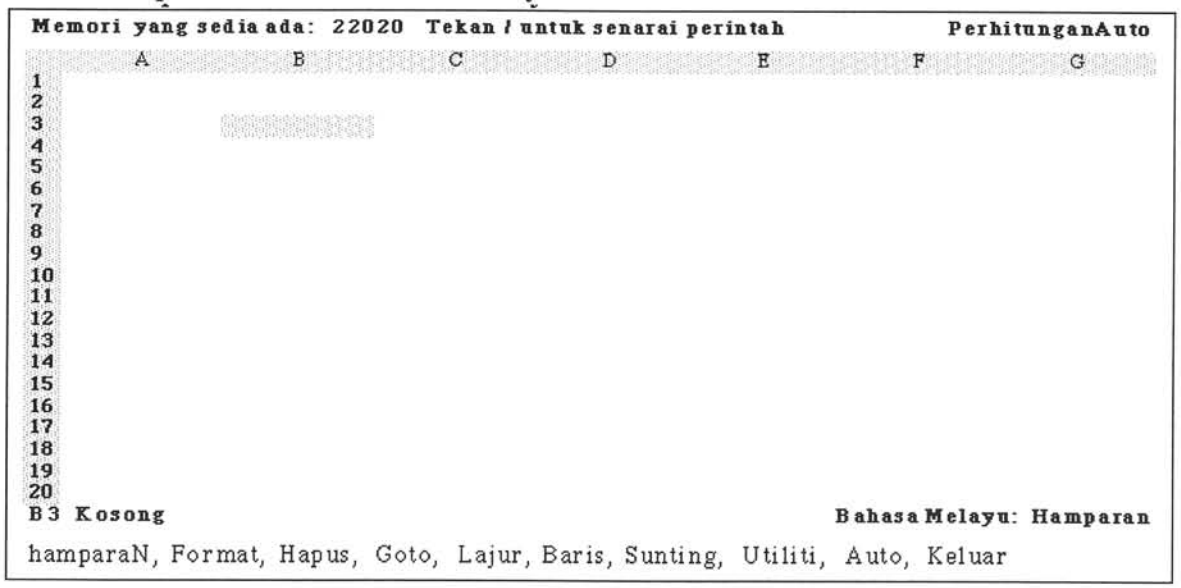
1 "Mäori: Te Ripanga"
2 "Patotöhia M kia Mäori"
3 "Käore e taea te whakatÜwhera i taua könae."
4 "Patotöhia / kia kitea ngä tono"
5 "Mahara wätea:"
6 "KUA HEE"
7 "Kua pau kë te wähi mahara"
8 "Kore kau"
9 "He Körero"
10 "He Nama"
11 "He Tätai"
12 "KaMahi Tätai"
13 "Whakatakotoranga"
14 "Tuhia te ingoa o te ripanga"
15 "Patotöhia tëtehi pätuhi kia haere tonu"
16 "Tuhia te whänui hou o te kapa"
17 "Kua whakatÜngia këtia taua könae. Me whakakore i te könae tawhito?"
18 "E kore e taea te katoa o tënei ripanga, ka pau kë te wähi mahara"
19 "Ehara tënei i te könae Ripanga FIRST"
20 "Käore kau he könae e mau ana i taua ingoa"
21 "Ka haere ki tëwhea pÜtau?"
22 "Tuhia tëtehi nama i waenganui i te %d me te %d"
23 "Käore e pai ana tënei nama pÜtau"
24 "Tuhia te pÜtau tuatahi hei whakarite"
25 "Tuhia te pÜtau whakamutunga hei whakarite"
26 "Kia örite ai te rärangi, te kapa ränei"
27 "Kia whakataka te pÜtau ki te taha katau?"
28 "Kia whakarite moni ngä nama?"
29 "He ..kei waenga i ngä nama?"
30 "Kia hia ngä mati whaïra?"
31 "Kia 132 ngä kapa i te mahi tä?"
32 "Ka tukua ki tëwhea könae (patotöhia te ENTER anake hei tä)"
33 "Me tä hoki i ngä pae pÜtau?"
34 "Taihoa, kei te whakatitea..."
35 "Taihoa, kei te tiaki..."
36 "Me tiaki i tënei ripanga?"
37 "Kua pokea e te rähi o ngä tono."
38 "riPanga,Wh'rite,wh'Ngaro,Haere, Kapa,Rärangi,wh'Tika,Mahi tätai,Ahua,wh'Oti"
39 "PWNHKRTMAO"
40 "Huaki, Tiaki, Mahi Tä, Whakangaro"
41 "HTMW"
42 "whakaUru, whakaKore, Whänui"
43 "UKW"
44 "whakaUru, whakaKore"
45 "UK"
46 "whakätu Tätai, whakätu Nama"
47 "TN"
48 "AK"
49 "$"

APPENDIX B

Screen Dump of FIRST in English



Screen Dump of FIRST in Bahasa Melayu



Screen Dump of FIRST in Mäori