Working Paper Series
ISSN 1170-487X

Strategies of Internationalisation and Localisation: A Postmodernist's Perspective

by Robert Barbour and Alvin Yeo

Working Paper 97/19
July 1997

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

Strategies of Internationalisation and Localisation: A Postmodernist's Perspective

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

Many software companies today are developing software not only for local consumption but for the rest of the world. We introduce the concepts of internationalisation and localisation and discuss some techniques using these processes. An examination of postmodern critique with respect to the software industry is also reported. In addition, we also feature our proposed internationalisation technique that was inspired by taking into account the researches of postmodern philosophers and mathematicians. As illustrated in our prototype, the technique empowers non-programmers to localise their own software. Further development of the technique and its implications on user interfaces and the future of software internationalisation and localisation are discussed.

Keywords

internationalisation, localisation, postmodern critique, user interfaces

1. Introduction

Recent developments in postmodern philosophy and critique have significantly changed the ways in which language and communication can be viewed. Against the background of these changes, the computing software industry is actively promoting the concepts of global information exchange standards. We take account of postmodern critique and demonstrate the possible impact of the critique on the software industry. Our examples are drawn from, and illustrated with reference to, a spreadsheet application.

Usually, software is supplied in English and frequently is also available in many of the major languages of the world, for example, Chinese, Arabic, French and German. Although there may be one software concept, such as a spreadsheet, there will often be a different version for each different language. Much research is still being conducted into ways to develop software so that there need only be one version of the software which can cater for all the diverse cultures. In this paper, we introduce the processes of internationalisation and localisation. We also highlight some of the strategies that might be taken to create software for the rest of the world while taking account of postmodern critique.

In Section 2, we will define our terms within the fields of critique and the international software industry. We outline facets of the postmodern critique in the form of a synthesis drawn from the literature. We will suggest reasons for the move towards internationalising software in Section 3. Section 4 will describe a number of current approaches to the problem of providing software for other cultures and languages. Our proposal for different internationalising strategies will be put forward in Section 5, and lastly, in Section 6, we will suggest some ideas about what the future holds for multicultural software using the processes of internationalisation and localisation.

2. Internationalisation and Localisation

Unicode (1995) defines software *internationalisation* as "the process of making a (computer) 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." In other words, internationalisation is the process whereby culture-specific components are isolated or extracted from the software. What remains is the software that is independent of culture-specific components such as language.

Software *localisation* refers to the modification of software to allow its use in a specific locale. A locale is a geographical region defined by a country, language or set of cultural conventions. A country might have more than one language in the locale. For example, in New Zealand, two main languages are spoken, English and Maori. Thus, software for the New Zealand locale would provide both an English language and a Maori language version.

Most of the earliest computer software was developed in North America, Britain and Europe by and for people in these countries. There was little demand for software outside the English speaking world. As the number of computers increased around the world, demand for software also grew. Economic growth, especially in Europe and Asia, probably contributed significantly to this demand (see Section 3). With this growth in overseas hardware and software markets, software developers began to provide versions of their original software for other localities. Some companies might also have been forced to seek these overseas markets to survive economically.

Early localised versions were probably direct translations of the original software. This approach would have been an expensive and labour-intensive process. To tailor the software to the target markets' requirements, the developers would need to accommodate the locales' character sets and sorting order, translate the user interface (text and icons) and documentation, such as manuals. Re-engineering or rewriting the source code would have been common requirements.
Software developers began to realise that if the software was initially designed to provide for their future markets' languages and cultural conventions, little or no modifications would be required. This approach would generate more savings and increase their revenue as software would reach customers faster, given less revision of code.

Much of the work involved in establishing and maintaining the internationalisation process is bound up in the activities of the United Nations Standards Working Groups such as ISO JTC1 SC22 WG20, the working group concerned with internationalisation.

Against a background of standards, making possible the mechanistic application of computing expertise in the service of people and communication, postmodern critique raises serious questions about the efficacy of the current global internationalisation project. We take 'postmodern critique' as a suitable label which encapsulates the research of a diverse group of philosophers, and other thinkers. This group examined and thought about the consequences of the Enlightenment in the so called 'modern' era. The first point in the critique emerges from the investigations of mathematicians (such as Brouwer (1907), in Heyting, 1975). This research has shown that any set of signs will suffice for expressing mathematical relationships. From this work, we draw the point that there is no one set of correct signs (Brouwer, in Heyting, 1975) or alphabet, rather there is whatever people have agreed on for particular purposes. In short, meanings and their associated signs come from use, not immutable truth (Wittgenstein, 1963). Taking account of this point of critique

would require software developers to ask the endusers to provide appropriate culturally sensitive interface details. Because cultures change how people use communication media, this aspect of localisation should be left entirely to people who are the endusers of the software. It is characteristic of the new western hegemony to assume that better solutions are those created in the West. Further support for a more tentative view of knowledge is found, particularly in the physical sciences and is derived from the Heisenberg's Uncertainty Principle (Heisenberg, 1959) which briefly says that the attributes of a measurement of a particle are always uncertain. The conclusion drawn from this understanding is that, if the sciences (which are in pursuit of truth about the physical world) find no certainty, then all other human activities are without certainty also. In the current context, Heisenberg's Uncertainty Principle draws attention to the essentially uncertain nature of physical human existence and to the role of the observer in human activities.

The catch-cry of 'Death of History' (Baudrillard in Grosz E.A., 1986) sums up the next item in the critique. Metaphorically, this label summarises the view that the historicity, assumed in western thinking and implying some form of necessary progression, is without acceptable evidence of support for any linear or other ordered pattern. From this position we argue that the changes that take place in information technology are not improvements simply as a consequence of being changes. Furthermore, the changes have no necessary outcome that is in any sense predictable or to be valued more than any other outcome.

The further point in the postmodern critique we wish to draw attention to is the 'Death of the Author' (Barthe, 1967). By making this pronouncement Barthe is drawing attention to the understanding that people do work on the text they attend to, such that whatever is understood from the text is the reader's creation from it, rather than the author's intentions. It is Barthe's view that the texts themselves cannot be taken as the authority because the meanings people create from them are forever open and undecided. Thus, there will be no one better-for-everyone way of building software, there will be no one interpretation from any one software interface or set of functionality. The only interpretation of a user interface that is of value to the user, is the interpretation which is understood by that user. Endusers in every country can expect that they will have software which is at least provided with the locally acceptable interface.

From Postmodern feminists, we draw attention to the point that the whole of the internationalisation genre is unconsciously gendered male, simply because gender and

gendered language is not reflected in the literature or the practice of constructing an interface to software. Women have been excluded because the language that might provide a gendered view does not exist in the internationalisation community. The postmodern critique resolves this state of affairs in the phrase: women as such do not exist but are in the process of becoming (Kristeva, in Nicholson, 1990). Should this be the case, then we can expect to see provision for gender related concepts in the various interfaces to internationalised application software. We think that software companies producing software which effectively provides for gender will have captured more than half the world market by that single step (all women and others not choosing to be gendered 'male').

Postmodern thinking observes that history is at an end with the arrival of liberal democracy as variously interpreted in the economies of the world. However, post modern critique also proposes that the global economy is under severe threat. This threat is seen in the fact that never have so many people been subjugated, starved and exterminated on the earth as is the case today. Postmodern critique points to the possibility of a similar decimation of the world's cultures. This state of affairs could arise where, for lack of digitised representation, in the form of agreed marks and signs, in the universal catalogue of human communications, a culture is isolated and written out of contemporary technological experience. Postmodern critique points to the leveling of cultures made likely by the 3 second sound bite mentality of media (Derrida, 1994). For many people of the most significant aspects of culture cannot be communicated in the three second sound bite and thus important differences in culture are excluded from the media.

The free market economy values profit before diversity. Cultural diversity reduces profit by spreading development initiatives over multiple markets. Internationalisation and localisation are the modernist responses to the poverty of multi-national marketing. Postmodern critique points to the contradiction in setting standards for signs used in communication since standards imply certainty (of which there is none) and consistency in sign use which, given the nature of human sign use, seems unlikely. We take the postmodern critique to highlight the following points:

- the signs (sounds, marks and gestures) people use to communicate are chosen and agreed by people
- there is no one correct set of signs or way of signing
- there is no necessary connection between any one set of signs and associated meanings

- both the signs and the meanings change as the use (people put them to) changes
- change in sign-use has no direction, signs are not better or worse for being changed
- culture matters, in so far as in it lie the agreements that people make about their signs
- people bring individually different histories to the signs they use and interpret
- signs marginalise individuals where aspects of signing neglect a culture either by failing to recognise that culture is more than signs or by marginalising the sign distinctions of particular cultures
- the free market economy as applied to the digital industries promotes the growth of single multi-national corporate entities which reduce product diversity in the interests of profit

We look at aspects of the process of providing software in the global context of a variety of cultural settings and then return to the issues raised by postmodern critique.

3. Why internationalise and localise?

Today, software developers are much more easily convinced to localise their products for markets outside the US. This change in thinking only happened recently. It was only in the early 1990s that there has been indication of growth in research into localisation of software (Russo and Boor, 1993),(del Galdo,1990),(Nielsen,1990). One reason that may have catalysed this development has been the economic boom in the South Asia region as well as the opening up of markets in China and Vietnam. The recent expansion of IT into these areas provides greater *economy of scale* in many sectors of the global economy. With greater economies of scale, more firms may find it financially feasible to localise software. A growing proportion of computer corporation's revenue has been derived from outside US , e.g. in IBM's 1994 Annual Report, 67% of IBM's revenue was derived from outside US. Furthermore, many more countries are adopting Information Technology into business, administrative and governmental organisations. Against the trend towards standardisation of software, the software market has become more culturally diverse with more countries becoming more technologically adept. The natives of many countries are beginning to, or have developed software or localised software concepts for their own consumption, e.g. the development of input and display of Chinese, Japanese and Korean characters.

Besides the monetary and efficiency gains, there are other reasons for localising software. When we localise software for small "indigenous" cultures, we are preserving elements of

that culture and indirectly preserving "a wider variety of perspectives, potential insights and solutions to the world's problems" (Griffiths,1994). This view is supported by a study carried out by the American Management Association (AMA) which reports that heterogeneous work groups generated work and business solutions that were more effective and innovative than homogenous work groups (HR Focus, 1993). If countries such as Japan were to succumb to the US software monopoly, their innovative input techniques may not have materialised.

From a social perspective, localisation of software can increase cooperation and communication between the countries involved. For example, software written in English and translated to Russian would require the collaboration and cooperation of people in the countries concerned. Software, such as CAL, requires large investments of time and talent. With localisation of such software, there would be increased world-wide availability of state-of-the-art software as software companies increased the distribution of software, thus meeting the needs of a wider range of people. Many projects fail due to lack of acceptance by target cultures. With localisation, such projects would stand a better chance of success. Furthermore, by localising software, communities less adept in software technology are exposed to state-of-the-art software. Thus, localising software could also be an effective mechanism for the international transfer of technology. We would also argue that the twin processes of internationalisation and localisation carry with them the consequence of increasing the hegemony of western cultures in the global software industry. We see the reasons for internationalising and localising software as a double edged sword for newly developing software industries of non-western cultures. One cutting edge of this trend is the provision of state-of-the-art software in an apparently useful form that facilitates trade and communication. The second cutting edge relates to the increased hegemony of western software manufacturers over the global software industry. Resolving these issues will require a resolution of the conflict between the claims for interdependent intercultural communication over the claims for cultural diversity and technological independence.

Experience shows that people would be more motivated to learn about IT if they were able to interact with computers in their own language (Griffiths,1994). In software that was localised, Bulgarian children attempted to write creatively and explore the software. But when given English software, the children only drew and painted.

Learning is also much faster if the user interface is more intuitive, that is, the interface makes use of users' existing knowledge. For example, a trash can icon is intuitive

recognised by Americans as a place to put things that are subsequent taken away. We argue that the unique understandings of people in a cultural cannot usefully appear in a standards table because of the diversity and ephemeral nature of human understanding However, we would also agree that the issues currently being addressed within the standards community will facilitate that exchange of data between cultures. Of course, such data exchange is essential to support the trade required for the global economy.

3.1 Ethnocentrism

The *ethnocentrism* of the software developers in the US is one reason the localisation process was not carried out until recently. The Oxford Dictionary defines ethnocentrism as the egotistical opinion that their own race is the most important. This ethnocentric behaviour can be described by this abstract from Gannon and Associates (1994):

> "Consequently, US economic success and military might often induce egotistical reactions to international event among its citizens. These narcissistic feelings are frequently evoked by news media's persistent portrayal of and fascination with the world disasters and mishaps. To the American media, "no news is good news," and thus the outside world is often reported as volatile, violent, and miserable. The positive features of foreign countries are rarely highlighted by US media, because the average American is constantly bombarded with news of famines, wars, violence, and political turmoil from the outside world. he or she is frequently not aware of the pleasant and fascinating features of other nations. This ethnocentrism is so extreme that many Americans believe that the United states is the safest and most prosperous country in the world, even when the facts do not support such a conclusion."

Such ethnocentric views may have prevented the widespread development of localised software. Other examples of this ethnocentric behaviour: rationalisation "they can understand English" was given for not translating a product's documentation and on-line help (Sprung, 1990); an American international marketing manager querying whether translation was at all necessary was quoted "because isn't 1992 when everyone in Europe is going to speak English?" (Sprung,1990).

4. Different Approaches to Internationalisation and Localisation

There are a number of ways to localise an application. We will use as an example, the localisation of a US English word processor to Maori (Te Ripanga, 1996) . The localisation can be carried out in three ways.

The first method is to modify the original application to accommodate Maori. In this case, some components of the original word processor will be altered. For instance, the word processor's user-interface-text might be changed to Maori and certain icons might also be altered. Thus, you would end up with a modified US word processor which has a Maori interface. However, if the word processor is to be converted to Arabic, the text and icon modification will not be sufficient as Arabic uses script characters. Furthermore, Arabic is bi-directional; the script characters are written from right to left, but numbers are written from left to right. In this case, we would use the second method to localise the word processor.

The second technique is to develop the word processor from scratch. This approach will take a long time, as one would need to go through the software development life cycle to create the Arabic word processor. Thus, you would end up with two word processors, one US English and one Arabic version.

The third method of localisation is to go through two phases; an internationalisation and then a localisation phase. This approach is carried out in the design stage where you remove all the culture-specific components (such as date, time, currency, number formats, and visuals), and ensure that the software will be able to accommodate all the languages, that is, Arabic, Maori and English. Thus, the internationalised component is devoid of culture-sensitive components. After the internationalisation process, the culture-specific components are "added" to the internationalised component in the localisation process.

In this third method, the internationalisation can take three forms; compile-time, link-time and run-time internationalisation (Taylor,1992). These different forms relate to how a piece of software is developed.

4.1 Software Creation

The origin of an application or software starts with a piece of text called the *source code*. This source code is normally a text file comprising instructions that dictate how the application will function. The source code is then *compiled* into an *object file*. The object file is an intermediate file between the source code and the executable, the final program that you run. The object file is a form of assembly language and is missing a number of components such as library functions that your program calls and start-up code. Thus, the object file must be combined with the start-up code and library functions in a process called

linking to get the executable. The final program, the executable file, usually has a extension *exe* at the end of it. Refer to **Error! Reference source not found.** below for an a summary of the compilation and linking process.

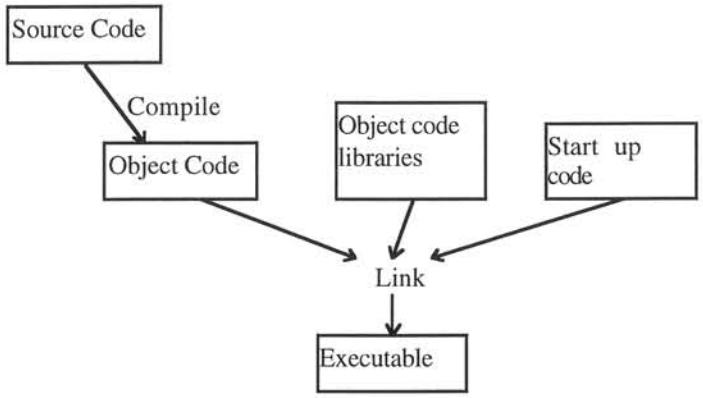

Figure 1: The program compilation and linking process

In compile-time internationalisation, the software is developed so that to localise the software, the addition of culture-specific components is carried out before the compilation process. In this case, the source code is modified, compiled, linked and run. Suppose there is a function in the software that prints out the word "Hello". To localise the function to French, the source code (text file) modification might include changing the text "Hello" to "Bon Jour". The file is then compiled, linked and when run "bon Jour" appears in the correct place(s).

In link-time internationalisation, the addition of the culture-specific components is carried out before the linking stage (that is after compilation). The culture-specific components are actually object files which are linked with culture-independent object code. For example, the object code to print the "Hello" message in Thai is linked with the internationalised code to obtain the Thai version; object code to print "Hello" in Greek is linked with the internationalised code to obtain the Greek version.

In run-time internationalisation, the addition of culture-specific components is done after the linking stage. In this case, the addition of culture-specific components is carried during run-time. The culture-specific components are placed in files. Mechanisms are added to allow the "culture" files to be selected. For example, in one file may contain the Chinese fonts and icons, and in another file, German fonts and icons. During run-time, you might wish to select the Chinese file, at other times you might want to use the German file. This technique is much more flexible as you do not need to conduct any compilation or linking in order to obtain different localised versions.

4.2 Other factors to be addressed

There are many factors that need to be addressed when we are internationalising and localising the software. Date, time and currency formats are now covered in current software such as Windows 95 and Macintosh Operating Systems. In these systems, you can select the formats that match your local setting. Other factors that should be looked at, include the accommodation of weekends (different because of religion), day turnovers, telephone and address formats, character sets and their collating/sorting order sequence, and reading and writing direction.

To some cultures, certain words seem harmless to them, though the same words might be downright offensive in another culture. Thus, translation of new words, names and menu accelerators needs to be conducted carefully. Colours, sounds and visuals can also be perceived differently by individuals from different locales. Details on treatment of these attributes are covered comprehensively in Fernandes (1995), Kano (1995), Chris Miller (1994), O'Donnell(1993), Apple (1992), Digital (1992), Taylor (1992) and Uren et al. (1993).

5. Our Proposal

In this section, we describe our research into two forms of internationalisation, the compile-time and run-time internationalisation. We then extended the results of the experiments to propose the token concept.

In order to experiment with the various forms of internationalisation, we developed a partially internationalised spreadsheet , FIRST (First Internationalised Research Spreadsheet Tool). FIRST was developed from TCALC, a DOS-based spreadsheet bundled with Borland C++. We obtained permission from Borland International to modify the spreadsheet for our research. TCALC is a fully functional spreadsheet which accepts text, numbers and formulas. TCALC also has common spreadsheet features such as load, save, print, insert and delete rows and columns, formula display, automatic recalculation and mathematical functions such as sin(), cos() and tan(). We chose TCALC as TCALC's source code was available. The availability of the source code meant that we did not have to start developing the spreadsheet from scratch.

5.1 Compile-Time Internationalisation

We first explored with compile-time internationalisation by extracting all language sensitive visual components, that is, the components that the users see and interact with. These language sensitive components include menu names, menu item names, status bar displays of cell contents (text, number, formula), dialog and error messages, instructions and commands.

These culture or language sensitive components (message tokens) were placed together in one location in the source code. These components were then translated into Maori. A Maori computer science lecturer carried out the translation. After the translation, the source code was then compiled, linked and run. As such, only one language was available after this type of internationalisation and localisation.

5.2 Run-time Internationalisation

We next experimented with run-time internationalisation. Since all these visual items are text-based, we placed these message tokens (language-sensitive components) into a text file. These text files are also called resource files in other literature (Kano (1995); Pugh and LaLonde,(1992)). Examples of the files in English and Bahasa Melayu (Malaysia's national language) can be found in Appendix A and Appendix B respectively.

Message Tokens in English	**Message Tokens in Bahasa Melayu**
...	...
14 "Enter the file name of the spreadsheet:"	14 "Taipkan nama fail untuk dimuatkan"
15 "Press any key to continue"	15 "Tekan mana-mana kekunci untuk menjalankan hamparan"
16 "Enter the new column width:"	16 "Taipkan lebar lajur baru:"
17 "The file exists. Do you want to overwrite it?"	17 "Fail tersebut wujud. Mahukah anda menulisgantikan fail lama?"
18 "Not enough memory for entire spreadsheet"	18 "Memori tidak mencukupi untuk seluruh hamparan"
...	...

Figure 2: Message tokens in English and Bahasa Melayu

In every text (language) file, each numbered line corresponds to the same message token but in a different language. For example in Figure 2, token 14 contains the dialog message "Enter the file name of the spreadsheet" which is equivalent to "Taipkan name fail untuk

dimuatkan" in Bahasa Melayu. (There are 49 tokens in total). Thus, when English is selected, message token 14 will be displayed in English. Likewise, if Bahasa Melayu is selected, then message token 14 will be displayed in Bahasa Melayu. In order to allow different languages to be used at the same time, functions were added to FIRST to allow the use of different text files (different languages) during run-time. Thus, one could select different languages at run-time without having to compile or link the software each time a new language is required. The selection of different languages is carried out by pressing certain key combinations, for example, ALT-E for English, ALT-B for Bahasa Melayu and ALT-M for Maori. Each time the key combination is used, all the message tokens (in a particular language) are loaded into memory and these message tokens are displayed on the screen. Thus, when ALT-E is pressed, all the text messages in English will be loaded as message tokens 1 to 49. When ALT-B is pressed, the text messages in Bahasa Melayu are loaded into message tokens 1 to 49 .

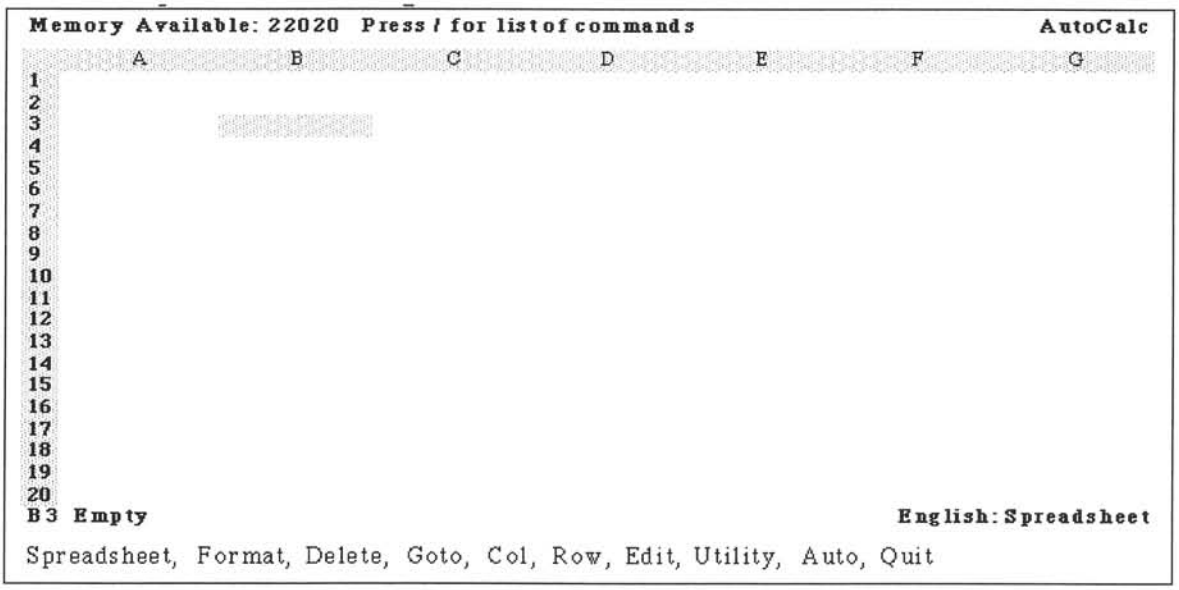

Figure 3: Screen Dump of FIRST in English

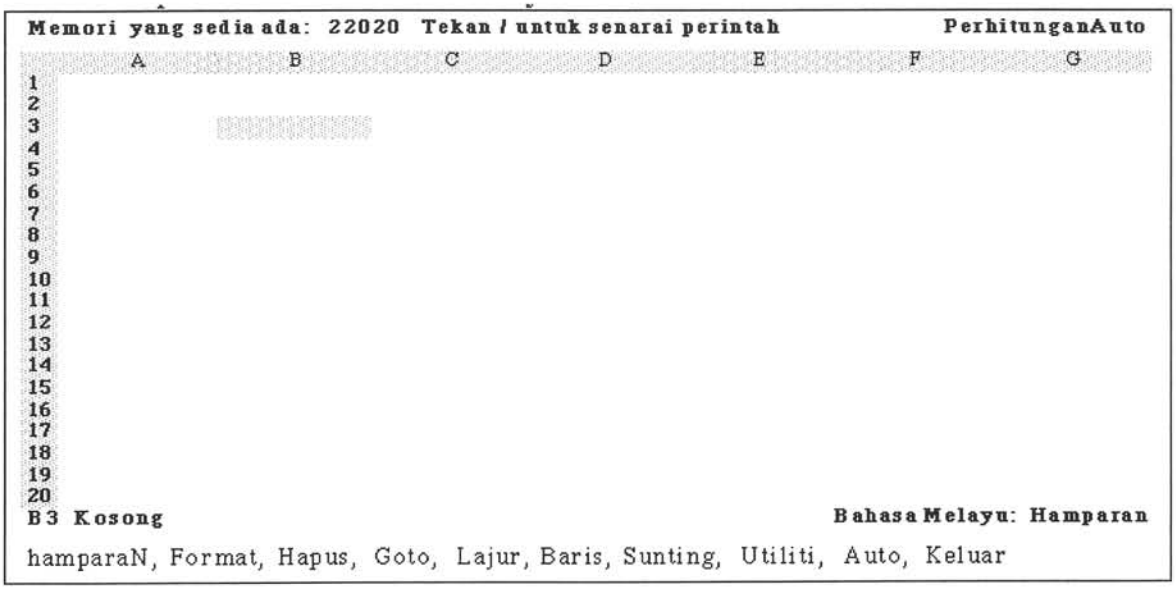

Figure 4: Screen Dump in Bahasa Melayu

All forty nine tokens in FIRST contain only text. This text can be interchanged with any other text provided the text is generated from Roman characters. Thus, to localise FIRST to other languages, a user has only to modify the message tokens in the text files. This feature allows a non-programmer to localise his or her own software. If this feature is incorporated into the software, smaller minority groups would be able to localise the software to their own culture and language. Thus, the survival of minority languages would be improved especially those languages which do not have the numbers to attract the attention of large computer firms.

Furthermore, the token concept can be used not only to encompass text. When we communicate with one another, we use one or a combination of these tokens; marks (e.g. text, Morse code, pictures), gestures (e.g. sign language, body language) and sounds (e.g. talking, sirens). We follow Brouwer (in Heyting, 1975) and believe that there is no token that has any intrinsic meaning in isolation. A token has no meaning unless it is interpreted by someone who already knows what it means. Thus, the token in the text file need not only be Latin -1 marks, as used in FIRST. We can extend the use of this token concept to include more complex marks like Chinese characters, sounds, pictures and gestures, if required. Thus, besides having files of tokens made of marks, we can also have sound tokens. With the inclusion of sound, computers could be accessible to the visually impaired. Of course, for an interface to accommodate marks and sound, certain mechanisms to map the sound with interaction methods would need to be created.

5.3 Proposal Summary

With our technique, tokens have no intrinsic meaning, only that which people agree they should have in use. Tokens are machine readable strings that distinguish between humanly readable sounds, marks, and gestures. In the FIRST prototype, tokens consisted of text. The FIRST software can accommodate languages that use Roman characters.
Our extended proposal is that beside text, these tokens can be mapped to contain sounds, and pictures. Also, in FIRST, unlike current resource files, a user without any programming knowledge can now make the necessary changes. Using these techniques also has advantages for programmers in that source code is protected and only executables are provided for users. The advantage for the software industry is that copyright is more easy to defend where software is released in machine readable form.

6. Future of internationalisation and localisation

We find that the internationalisation/localisation process contains certain contradictions which we have brought into sharper focus by using postmodern critique to pick out key aspects of potential change in the global software market. We think the issues of international trade and communication are crucial for the continued peace and prosperity of the world's people. Against this need for global communication is set the specific needs of individual cultures to preserve, promote and grow the particular solutions each culture finds to the problems of community living and communication. The necessary tension between single international providers of internationalisation and localisation standards and processes, and local software developers seeking to articulate particular cultural views of communication, should be promoted for the survival value it offers the world's people. We would argue for active steps to be taken to promote indigenous software development by preserving the markets for expressing monetary exchanges in a bilingual form so that each culture is required to provide source and target software versions. In such a regime the trading partners would undertake not to swamp local markets in localised versions of other than core applications software for supporting financial transactions. Rather, we would prefer software suppliers to develop codes of practice to help them meet their obligation to preserve cultural diversity by training and employing indigenous people to act as implementers of locally created software concepts. Addressing these issues will require software that facilitates data exchange between nations. We have attempted to provide a model for ways in which this process might be expedited with FIRST, our customisable multilingual spreadsheet for Latin-character-using-nations. Lastly, we would argue that taking into account the contradictions emergent in postmodern critique serve to highlight

directions in which software could develop. From our point of view, the most important outcome of software which accommodates postmodern critique, is growth in respect of the contribution minority cultures could make to global communication. This contribution will be found in the types of innovative solutions exemplified by Japanese input mechanisms. We look forward to a continued diversity of software solutions which is a consequence of valuing cultural difference.

7. References

1. Apple. 1992. *Guide to Macintosh Software Localization.* Reading, Mass. Addison Wesley.
2. Barbour, R.H. 1996. Te Ripanga, the spreadsheet. Proceedings of the New Zealand Computers in Education Society 6th Biennial Conference, Hamilton New Zealand.
3. Barthe, R. 1967. Image, music, text; essays selected and translated [from the French] by Stephen Heath. London : Fontana, London 1977.
4. Baudrillard, J. 1986. *The year 200 will not take place*" In Grosz et al., 1986 Futur*Fall: Excursions into Postmodernity. Power Institute of Fine Arts Sydney.
5. Chris Miller L. 1994. Transborders Tips and Traps. *BYTE.* Vol 19. No 6. p93-102.
6. del Galdo, E. 1990. *Internationalization and Translation: Some Guidelines for the Design on Human Computer Interfaces.* In Jakob Nielsen (Ed.) Designing User Interfaces for International Use. Elsevier, New York. p1-10.
7. Derrida, J. 1994. *Specters of Marx : the state of the debt, the work of mourning, and the New international.*: Routledge, New York,1994
8. Digital Equipment Corporation. 1992. *Digital Guide to Developing International Software.* Digital Press.
9. Fernandes, T. 1995. *Global Interface Design.* AP Professional.
10. Gannon M J and Associates. 1994. *Understanding Global Cultures: Metaphorical Journeys through 17 countries.* Sage Publications.
11. Griffiths D, Heppell S, Millwood R, Mladenova G. Translating Software: What It Means and What It Costs for Small Cultures and Large Cultures. *Computers in Education.* Vol. 22. No. 1/2. P9-17.

12. Heisenberg W. 1959. Physics and philosophy : the revolution in modern science. London : Allen & Unwin.

13. Heyting A 1975. Brouwer L. E. J. *Collected Works 1 Philosophy and Foundations of Mathematics.* Amsterdam : North-Holland Pub. Co. ; New York : American

14. HR Focus. 1993. More Companies Are Drawing Strength from Diversity. *HR Focus.* Vol. 70. No 6. p2.

15. IBM Annual Report, 1994.

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

17. Nicholson L.J. 1990. Feminism/postmodernism New York : Routledge.

18. Nielsen J. 1990.*Usability Testing of International Interfaces.* In Jakob Nielsen (Ed.) Designing User interfaces for International Use. Elsevier, New York. p39-44.

19. Nielsen J (Ed.). 1990. Designing User Interfaces for International Use. Elsevier, New York.

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

21. Pugh J, LaLonde W. 1992. Internationalizing your application. *Journal of Object oriented Programming.* Vol. 4 No. 8. January 1992. p59-62.

22. Russo P, 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. p342-347.

23. Sprung R C. *Two faces of America: Polyglot and Tongue-tied.* In Jakob Nielsen (Ed.) Designing User interfaces for International Use. Elsevier, New York. p71-102.

24. Taylor D. 1992. *Global Software: Developing Applications for the International Market.* Springer-Verlag.

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

26. Unicode. Unicode and Internationalisation Glossary. http://www.stonehand.com/unicode/glossary.html. 1995.

27. Wittgenstein L. 1963. *Philosophical investigations* translated by G.E.M. Anscombe. Oxford : Blackwell

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 "$"

Appendix B

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"