Working Paper Series
ISSN 1170-487X

Character-less Programming II: the Spreadsheet

by: Robert H Barbour

Working Paper 95/30

November 1995

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

Character-less Programming II: the Spreadsheet

Robert H Barbour

Department of Computer Science, University of Waikato, Hamilton New Zealand

Abstract

The spreadsheet is a commonly used, yet under-researched, application tool. Hendry and Green (1994) report fewer than ten entries about spreadsheets in the HCI community literature between 1984 and 1991. The 1993 Human Computer Interaction Proceedings of a Conference on Applications and Case Studies does not list the word 'spreadsheet' in the index whereas database and wordprocessor are both referenced. The current situation in 1996 is not significantly different. For Polytechnic and University communities, many of whom are involved in teaching about application software, this situation means that the widespread teaching of the tool is simply not being reported in the research literature.

The unanswered but researchable questions raised in the Hendry and Green (1994) indicate the breadth of the field. Following a summary of these questions, the even more neglected issue of providing culturally appropriate spreadsheet software is raised. A set of constructs for thinking about multicultural software is presented. A model of a procedure for generating an elementary spreadsheet is provided, exemplified and demonstrated for the Mäori Language in New Zealand.

Introduction: the Ecology of a Research Area

The spreadsheet has been much used as an application tool since Visicalc appeared in 1978. The tool was so influential that Stern and Stern (1993) argue it lead people to purchase machines, such as the Apple PC, that ran the software. Hendry and Green (1994) state that low level and higher level questions could be raised about the spreadsheet as an application. Their questions are suggestive of studies that could be conducted into interaction tasks such as data entry and formulae creation. Careful examination could be made of the part the tool plays in the user's life, of the other computer based tools spreadsheet users employ and the relationship between tools. Little is reported about how spreadsheet users construct the models they use, edit them according changing circumstance and eventually discard them when outdated. It is not known whether there is a hierarchy of spreadsheet users with some people acting as producers of templates while others are consumers. The work of Nardi et al., is specifically mentioned as being the only substantive body of research into the wider aspects of spreadsheet use. In summary, Hendry and Green (1994, p1035) state that

> 'field studies show that spreadsheets are an effective communication tool and that the computations performed are generally simple'.

They characterise the research literature as 'dominated by bedazzlement'.

Three key aspects are selected as amenable to research. Hendry and Green (1994, p1038) ask:

- how usable is the formula language?
- what evaluation has been carried out on the spreadsheet as a 'program'
- how effective are spreadsheets as interpersonal communication tools.

My view is that research into the teaching of spreadsheet programming and use is one of computer science education's 'green field' areas. I hold this view because there is very little, as yet, known or reported of the ecology of the spreadsheet. In explanation for this situation I note that software is, almost without exception, distributed as executable code. Extending such software is difficult and permissions to obtain and modify source code are unheard of in the software industry.

Consequently, it is very difficult to set up a system to monitor the things a person does with a spreadsheet or any other application (Barbour, Ford, Cunningham, 1993). People who might be interested in studying the way spreadsheets are used (Yeo, 1993) find insurmountable difficulties because the software cannot be extended to ensure that the application becomes a research environment as well as a useful tool.

Among the minimum facilities a research application could include are:

- the functionality of a common application such as a wordprocessor
- a file system which provides mechanisms for isolating one user from the next and controls user access to files on the system by some form of transparent file security.
- a record and playback feature which logs all states, events procedures and process.
- a file pre-processor which identifies cursor and mouse movement
- a facility for extracting dialogue by event type from a user's file.

An application with these features would encourage active research and reporting about the teaching of that application. So far as I am aware there are no commercially available applications which meet these requirements. As a general observation, the discipline of computer science rarely uses computer based tools as a basis for studying its own teaching/learning activities. Using computing tools for enhancing the process of

teaching/learning about the discipline seems an ideal place in which to test the claims that are made for the computer in educational settings (

A model for thinking about Application Software

The interaction that takes place between a person and a machine can be viewed as an interaction between two cultures: the culture of the programmer and that of the enduser. If the programmer is also the enduser then the cultures intersect exactly. The problem of communication is simplified but not unimportant. Within a culture there are differences in the way people learn and go about the business of communicating which can inhibit the communication process (Pask, 1976, Schmeck, 1988). However, in most cases the culture of the enduser is more or less distanced from that of the programmer. Speaking the same mother tongue helps reduce confusion but, since each act of communication with a machine requires bridging the gap between language as it is spoken and language as it is represented, confusions can still arise.

Reducing confusion in a human computer interaction requires an intersection of cultures in the form of a common communication medium. The necessity of creating this intersection is one of the most significant problems facing a programmer. It is increasingly evident that users no longer tolerate poor interactions with software. In the sense used here a poor interaction is one during which the nature of the interface prevents the enduser from achieving a satisfactory outcome from an interaction with a machine.

It was argued in Hopper and Barbour (1994) that the culturally-dependent aspects of an interaction such as language and character set, can be separated from the process of translating programs and from the execution of a translated program.

Expression of Meaning

All human languages use the notion of 'a mark or sound in a shared context' to enable people to determine the meaning of an utterance. This assertion applies whether two friends are in idle conversation or a programmer is constructing a compiler.

The context is provided in three parts :

- That part which is shared in the common milieu of those communicating.
  For example, when using the English word 'drive', the two friends in conversation in New Zealand would probably be referring to an access way from a road to a building. A programmer communicating with a computer almost certainly refers to a sequence of marks used to identify a file system storage device.

- That part which is composed of sounds or marks from which the recipient creates a meaning.

- That part which is contained in the structure of the utterance itself , the order in which marks or sounds are presented (the syntax of the language).

The marks, made visible as the word 'drive' are therefore accepted as having no intrinsic meaning. All English speakers who are computer literate, however, have an internal map from visible (audible) token to (at least) the two meanings-in-context given as examples. Without these shared meanings-in-context communication is not possible.

Token Abstraction

It is well understood that a mark or sound (an utterance) has no intrinsic meaning in isolation. This concept seems not to have been applied in the design and implementation of computer based applications. The situation also reflects the natural attitude (Wagner, 1970) of programmers who are working within a single culture. A programmer would

take for granted that other people would share both context and thus the meaning of the marks and sounds used, without need for further communication.

The first computer language to make use of the idea of 'token abstraction' was APL, probably because of the richness of the visible forms of the marks used to express it. No computer character encoding used at the time of its development had the necessary richness of variety of visible marks. The latest version of this language, Extended APL (DIS 13751), follows its predecessor by naming tokens rather than giving them explicit visible expression in the standard (although, naturally, the names themselves are visible when printed). Of course, a name is not the things it names.

An implementation of APL requires a mapping from some external representation of a mark to the language-specified token. No functional distinction is made, however, between the two conceptually different things in the language implementation. This means that the programming language APL can be localised to any set of humanly readable marks of sufficient complexity to map from the APL tokens to the human language.

Language Tokens and Representation

Programming in the future may involve some design tool which will generate the syntactic tokens of a programming language. This development will be a contrast from the current situation where a string of marks is derived from a visible character representation. Thus, the first step in programming will be to code the functionality required in an application in the form of a series of tokens. The second step will require converting those tokens into a humanly readable set of sounds and marks for a specific target culture. It is this notion, together with those ideas taken from APL, Ada and Modula-2 language definitions which has lead to the proposed solution to the problem of providing multicultural software. Given that a programming language translator is passed a stream of syntax tokens in whatever internal representation it requires then it can subsequently be provided with the necessary humanly readable marks required by a particular target culture.

The internal representation of the syntax tokens is a simple task requiring a one to mapping of short sequences of marks into a token. The encoding of the marks is irrelevant as long as a mapping is provided to the translator. What is needed, therefore, is a list of all of those message tokens which are required by an application. A typical list would include tokens in classes such as: context-dependant-help tokens, End-of-file-token, End-of-Line token, White-space-token, Procedure-token and Quote-mark. The list, apart from being ordered in the language specification must, contain ALL the tokens that could be specified by applications. It is necessary to distinguish tokens in this list from those specified in an individual application, in which programmer-defined tokens may be divided into the following classes :

- Compositions of language defined tokens. Numeric values and for languages in which such compositions are needed, a list of language tokens will necessarily include such additional language-defined tokens such as Digit-zero, Digit-one, and the Exponent-mark.

- Bracketed data. For such data values a language need only define one or more marks to provide the brackets, even a single mark to 'toggle' between program source and data -- as, for example the Quote-mark as used in this sentence. Many existing language definitions identify strings of visible characters in this way.

- All other programmer-defined tokens

Provided the mapping of the external representations for language tokens then it has no need to be aware of the external form of representation of ANY token at all. It has no need to be aware of any character encodings used in its host environment -- for example any use of codes which a suitable representation engine may produce such as Bengali, English, Chinese or any other humanly readable/audible language 'words' or' digits' or 'punctuation'. A mapping provided for a translator in this way should be made available in the environment in which it is to execute, as part of the culture-dependent components of the host operating system.

An application meeting these requirements could be described as being internationalised, that is, easily capable of being localised for (almost) any cultural environment (ISO/IEC DTR 11017-1995(E) WD7C). A localised application is an adapted internationalised application fitted for a specific cultural environment. Thus, the semantics are preserved while the syntax may be changed. Each target culture could provide a set of localised elements stored in a table. Among the culturally specific attributes that may be included in language specific tables of sounds and marks are:

- character ordering sequence for strings
- the script the visible characters used in a written language
- writing direction
- multiple forms of characters, uppercase lower case
- hyphenation, punctuation, spacing
- expression of words as numbers
- classification of characters such as alpha and numeric
- messages in humanly readable form
- text length varies across languages
- spelling, color or colour
- documentation in users language
- text layout on the page
- character, glyph size and form, line size and line spacing
- font requirements
- provision of boxes in text
- icons and symbols
- mathematical symbols
- paper sizes
- data input methods
- ideographic scripts
- script and culture
- voice messages
- date time and calendar

- number format
- number rounding
- currency symbols
- price expression + or - taxes
- telephone number format
- postal address format
- measurement system
- colour significance
- function names
- personal titles
- taboo items/words
- regulatory requirements
- any other feature of a culture that enhances or inhibits the adoption of computer based tools

The technique described, therefore, enables the initialisation phase of a translator to read such a mapping in order to initialise its interface presentation system. During subsequent translation of some unit or program, the encodings received in the source stream are matched against the mappings and further presentation actions taken as required.

Data Manipulation

The suggested language-defined translator map completely specifies a character-less (and hence considerably more portable and culturally-sensitive) translator mechanism. However, it does not, of itself, solve the allied problem of data manipulation by an executing program.

Where data appeared in the source of a program, the translator lexical analyser merely collected encodings in a totally transparent manner. After all, it is the manipulation of such unknown data, together with that read from some input channel during processing, which is the principal purpose of a program. The production of data through some

output channel is only of related concern if it is intended for use by some other tool (eg a rendering engine producing sounds or visible marks for human users).

The encoding of data, whether as part of some program source or as data obtained/ generated during program execution is therefore of major concern to the writer of a program, but only insofar as the actual encoding employed may be 'understood' by the program. The solution to this problem given above in the case of a programming language translator (which is, after all, only another program), relied upon providing a mapping from encodings to the presentation marks and sounds in a completely portable and transparent manner.

The ideal would be for the specification of every program to indicate input/output languages as described above but this is unlikely to be practical for some time to come (if ever). As an interim solution, therefore, it will be necessary to provide a simple, practical mechanism which will improve upon the present situation -- i.e. it must be possible to determine the incoming tokens from the external representation.

Are Characters Really Necessary?

Characters are necessary for the following :

- Expressing in visible form in program source some complete or partial message for export to a human (or other text reading entity)-- e.g. "syntax error!".
- For use in converting to/from some internal token or value (converting a number or date, say).
- Either individually or in combination as a substitute syntax token (in much the same way as programs are currently written, just as some form of textual data).

The relation of these three uses to comments/data, language tokens and value composition as specified for programming languages is obvious. What, perhaps, is not so obvious is that these items have essentially all been eliminated from application programs which :

- Provide for 'messages' to be external as a list of messages in the same order as some internal enumeration. Naturally this is done to permit the use of different natural languages and orthographic forms of message.

- Use culturally shareable tokens which embody conversion analyser/generator code.

- Merely use encoding pattern matching to determine whether or not some encoding sequence matches a known syntax token. This technique uses encodings but not characters. The only point where characters are actually needed, therefore, is as visible forms (where an encoding uses a suitable rendering engine) for human interpretation/generation. This situation could occur, for example, where a human user is preparing a list of messages for some program to use

In essence, programming does not necessarily involves characters. In restricted circumstances, detailed manipulation of character encodings may be required. This is almost exclusively restricted to the use and preparation of mappings to those tokens which have some representation internal to a using program which is not the concern of any external entity.

Words and Strings

It is proposed, as a final part of the interim solution using tokens suggested earlier, that two very special tokens are mandatory -- Word and String, where a String is defined as a sequence of words separated by white space encodings.

From the point of view of the human user, the ability of a computer to organise visible text in accordance with some culturally appropriate rules is of paramount importance. The organised list of names, etc. is a very important feature of many human activities. These two tokens (and others which may be derived from them) are the only ones which necessarily use the concept of character as understood by the human user in the semantics of their operations, which include among other things, ordering and equality/inequality testing.

An exemplar

There is very little software that can accommodate the presentation requirements of the Mäori language in the user interface and font presentation system. Yet being able to

express oneself in the mother tongue is increasingly critical for current computer users. Making Mäori language available to speakers of the language requires the creation of specific software using a high level language or the modification of existing software. Since the objective is to provide a working version of a spreadsheet application creation from definition, through to coding of the spreadsheet seemed to miss the point. The spreadsheet should look and perform in the same way as a conventional spreadsheet. Thus, a search was begun for source code for a commercial spreadsheet program. It is possible to make extensive changes to Macintosh applications using a file editor such as **ResEdit** or **Resource Editor.** Unless the person making the changes is highly skilled, the likelihood of seriously damaging the functionality of the software during the 'hacking process' is quite high. No current commercial spreadsheet application is provided with the source at the point of sale. The task of reverse engineering a spreadsheet program is daunting and could, if successful, result in 'look and feel' legal actions. Source code was eventually located for both a Unix and IBM PC spreadsheets. Modifications were made to the PC software source files so that experiments could be made with the screen menus and message system.

- The first step involved re-crafting the code by converting print statements into message tokens. The spreadsheet is displayed top and bottom left in Figure 1.

- The second step was to convert the message tokens to the appropriate strings in the target language, in this case, Mäori. A computer literate speaker of the Mäori language provided the translation of the messages (Keegan, pers. com.).

- Thirdly, the message tokens were integrated with the source code as a 'header file' and screened in a prototype form (top right Figure 1).

- The fourth step was to modify the screen messages so that there is a fit between the message space and the message string (bottom right Figure 1).

- The final stage was testing the eventual spreadsheet as a running piece of code

Clearly, the resulting software is still monolingual and localised to the Mäori language. A truly bilingual application would have the facility to provide real time switching between screen displays of both Mäori and English commands and menus.

Providing an internationalised computer based application is complex but possible where the intersection between source and target cultures is large. On the other hand, where there are large discrepancies between cultures, such as in the fundamental structure of language or concepts, the localisation process may present major problems.

It is emphasised that the task of providing application software and computer technology in educational contexts across cultures is difficult. Minimally, a team must be set up consisting of individuals who are: literate in a disciplinary area and in computer science , and who are conversant with the cultural and language of source and target cultures, who are also aware of the mechanisms that can be employed to facilitate the internationalisation process and then capable of following up with the localisation process. These are non-trivial but essential requirements for people promoting computer supported internationalised educational activities.

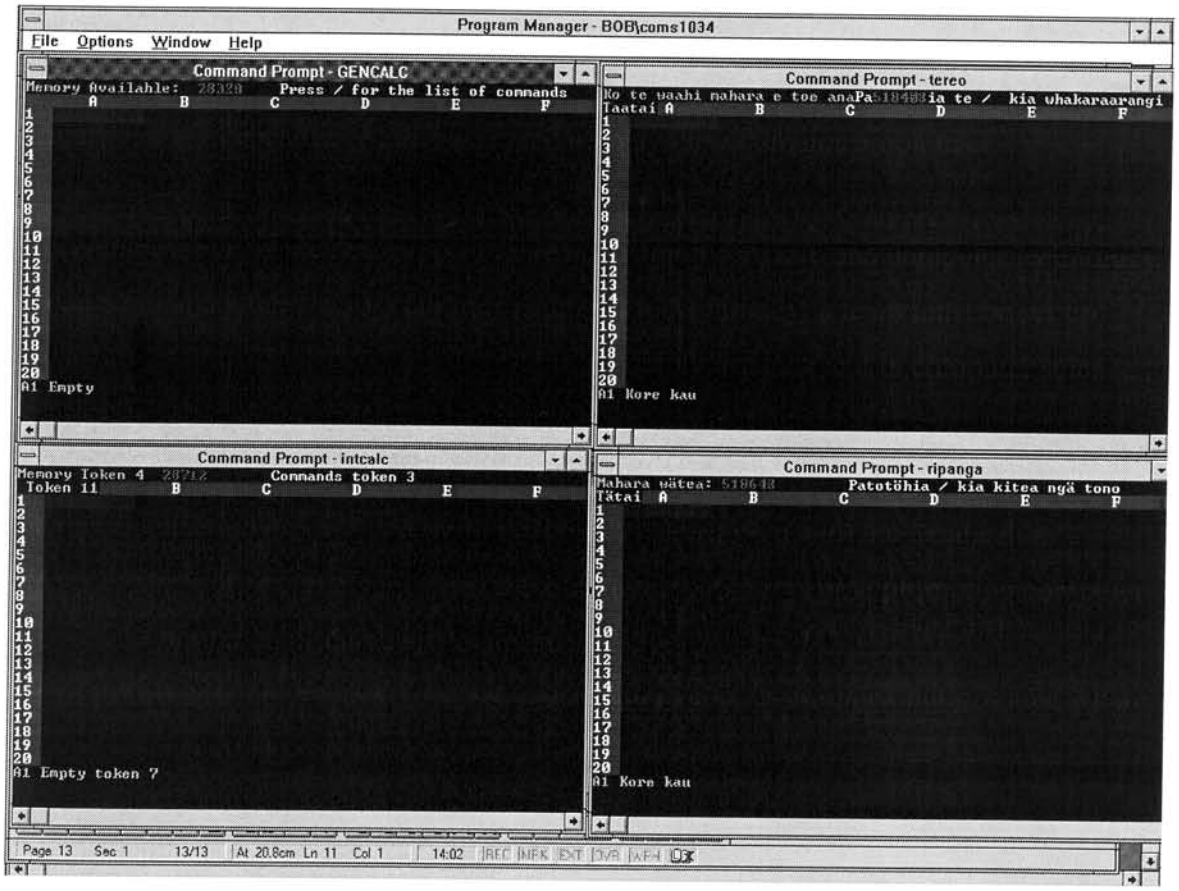

Figure 1 The development of Ripanga. **Gencalc** is a generic spreadsheet. **Intcalc** is a internationalised spreadsheet. **TeReo** is a first cut prototype localised spreadsheet. **Ripanga** is a second cut prototype spreadsheet..

References

Barbour R.H., S.J. Cunningham and G.Ford. 1993: Maori wordprocessor for indigenous New Zealand young children. *British Journal of Educational Technology*. 24, No.1 114-124.

Hendry D.G. and R.G. Green 1994: Creating, comprehending and explaining spreadsheets: a cognitive interpretation of what the discretionary users think of the spreadsheet model. *International Journal of Human Computer Studies*. 40 1033-1065.

Hopper K and Barbour R.H. 1994: Characterless Programming: Technical Report to WG20 Internationalisation of Application software. Tech Report (94/1) Department of Computer Science, University of Waikato, Hamilton New Zealand.

Keegan, Te Taka 1995: Personal Communication.

Pask G 1976: *Conversation Theory: Applications in Epistemology and Education.* Elsevier, Amsterdam.

Schmeck, R.R. (ed) 1988: *Learning Strategies and Styles*. Plenum Press, New York.

Stern N. and R.A Stern 1993: *Computing in the Information Age*. John Wiley and Sons, New York.

Wagner H.R. (ed) 1970: Alfred Schutz: On Phenomenology and Social Relations. University of Chicago Press, Chicago.

Yeo A. 1992: *Detecting Learning Strategies of First Year Computer Science students in a Spreadsheet Tutorial Environment*. Unpublished Undergraduate Report of an Investigation. Department of Computer Science, University of Waikato.

_ 1995: ISO/IEC JTC1 SC22/WG20 Internationalisation DTR 11070: Framework for internationalisation. Draft document.