TRANSLATING INTERACTIVE COMPUTER DIALOGUES FROM IDEOGRAPHIC TO ALPHABETIC LANGUAGES

Ian H. Witten
Man-Machine Systems Laboratory
Department of Electrical Engineering Science
University of Essex
Colchester CO4 3SQ
U.K.

Summary

A scheme is described which enables users to interact with a timesharing computer in an ideographic language such as Chinese. The host computer runs a completely unmodified English-based operating system, the necessary translation being performed by a preprocessing microcomputer which constitutes the ideographic terminal. The two systems communicate through a serial line, and the host sees the preprocessor as an ordinary (English) terminal. Although the examples are all drawn from Chinese, the scheme could equally well serve other ideographic languages like Japanese and Korean.

After brief consideration of the terminal's keyboard and display, we examine the software problems of translation for six example subsystems — two interpreted programming languages, text editor, document preparation system, interactive database management system, and the command level of the operating system itself.

The investigation shows that few limitations need be placed on the user's actions. For example, a suitable transliteration to alphabetics allows ideographic filenames to be used and manipulated quite naturally by operating system commands (which are themselved transliterated). The same transliteration allows ideographic text to be entered, edited, stored, displayed, and printed without difficulty. If text to be altered is located and specified by context (as is common with most modern text editors), rather than by character offsets, the transliteration must be designed to yield unambiguous pattern matching. Existing document-preparation software can be used for ideographic text, although there is a difficulty if English and ideographics are interspersed within the same document. The most difficult subsystem to handle is the BASIC language, with its baroque syntax and non-contextual string-matching operations: however, the paper shows how suitable action by the preprocessor can overcome all problems of standard BASIC except for operations which depend explicitly on ASCII codes.

Background

Recent advances in technology enable considerable improvements to be made over previous attempts to construct an ideographic terminal.[1,2] High-quality raster-based graphics systems provide an economical means of output. Hard copy can be obtained with high-resolution matrix printers[3] and laser-driven phototypesetters.[4] Character generation requires a fast read-only store for frequently-occurring ideographs, backed up by a slower store for the rest — this approach is encouraged by language statistics.[5,6] Furthermore, large read-only memories will soon become available which are suitable for storing picturegrams of thousands of characters. For example, at the recent International Solid-State Circuits conference in San Francisco, NTT's Musashino Electrical Communication Laboratory announced a 4 Mbit read-only memory, arranged as two duplicate halves of 2 Mbit each to reduce the effect of single-bit errors.[7] This could accomodate up to 8000 picturegrams, each stored as a 16 x 16 dot matrix. The major computer companies in the West have no product lines which include ideographic terminals, although IBM recently introduced a Katakana VDU.[8] However, their appearance can only be a matter of time.

Input of ideographic text is somewhat more difficult than output. Chinese typewriters have a table-sized keyboard with 2400 ideographs, supplemented by a library of extra characters which are inserted into the machine when required. The problems of inexperienced users with such a large keyboard are exacerbated by the difficulty of finding a particular character, for ideographic languages have no simple and universally-accepted "alphabetic ordering". Although novel keyboards and computer-based retrieval methods have been designed,[9,1] they do not appear to have met any great degree of acceptance.

However, a new scheme which employs 238 keys shows promise as a computer input device for Chinese ideographs.[10] The method originally covered a basic lexicon of 3260 characters, but has recently been extended to over twice this number without increasing the keyboard size. It is based on an analysis of characters into radicals, which corresponds to the way that Chinese children are taught to write in school. With 346 radicals, between 1 and 6 keystrokes are required for each character, the average being 2.7. (Compare this with the average of 6

alphabetic characters per English word.*) These 346 radicals are placed on the 238 keys in such a way that although some keys correspond to two radicals, no ambiguity is created when actual Chinese words are entered because of restrictions on valid combinations of radicals. Note incidentally that this simple technique has been used in situations where alphabetic English words are keyed on a 12-key touch-tone telephone pad.[11]

Taking these considerations into account, we envisage an ideographic computer terminal comprising a small microcomputer system with keyboard, display, and hardware character generator, possibly supplemented by a writeable character store which is maintained by the processor from a floppy disk holding the remaining, rarer, characters.

Given such a microcomputer-based ideographic terminal, it is clearly possible to write a specialized operating system and application software for it on a general-purpose timesharing system. However, the effort required is enormous, and the approach denies rapid access to new computer developments in the West. This paper explores an alternative, namely, the use of the terminal as a front-end processor for accessing English computer systems. Such a scheme is not without its disadvantages, for much data-processing depends upon an alphabetic structure of language. However, we presuppose no modification whatever of the host computer software and examine the extent to which the scheme can be made to work, and what facilities must be forfeit.

Semiotics of interactive computer systems

When users speaking other alphabetic languages, like French or German, interact with English-based computer systems, relatively few problems arise. Much interactive dialogue involves passive text, which is not interpreted or constrained by the system at all. For example, files can be edited or used for data-base retrieval, and comments can be included in programming languages, irrespective of the particular language used — providing it is alphabetic. String decomposition by character position, as is common in older text editors, creates no problem. Syntactic constraints are placed by the system on certain signs, which we call neutral. Examples are file names and other identifiers. However, there is no reason to believe that this causes any more difficulty in other alphabetic languages than it does in English. Symbols for constructs which have semantic import in the system, like keywords in programming languages and operating system commands, constitute active signs which must agree exactly with the representation stored internally in the computer system. Often, foreign users are content to write keywords in English, because of its predominance in the computer world — international computer language standards usually use English, most programming-language primers are written in English, and so on. However, altering the keyword representations stored in the computer system is not a difficult task, at least in theory: they will appear as character strings in the source code and editing and recompiling it is all that is necessary. Hence the issue of language-translation preprocessors which is tackled in this paper does not arise for alphabetic-language users.

With ideographs, the situation is not so simple. Certainly an ideographic computer terminal could interface to a standard serial line, emitting ASCII code sequences which represent ideographs and accepting them to generate an ideograph on the screen. Then the operating system tables which store keywords internally could be adjusted to hold the ASCII sequences which represent the particular ideographs used as keywords. Although this copes correctly with active signs, difficulties would still arise with passive and neutral ones. For example, string editing based on character positions would fail, unless the user was keenly aware of the internal representation of ideographs. It would be difficult to accomodate the syntactic constraints for neutral signs. BASIC identifiers, for example, must contain at most two characters, the first being a letter and the second, if any, a digit. Resolution of these problems requires a more radical modification of the system software than simply altering tables. Another, simpler, alternative is to leave the system unchanged and to try to incorporate the necessary intelligence in a preprocessor.

The kind of signs that must be dealt with, then, are as follows.

Passive
These constitute text that is not interpreted by the system. Certain special characters, which are interpreted at device driver level — like interrupt signals, rubouts, and line-erase characters — must not appear by accident in the passive text. The possibility of editing by character offset must be noted.

Neutral
These symbols must conform to syntactic constraints imposed by the system. Examples are
filenames
identifiers in programming languages.
Most systems insist that neutral symbols contain only printing characters (and not control

*Of course, the information in a single keystroke is much higher in the Chinese case. Using a naive zero-order approximation which does not take into account character or radical frequencies, the entropy of a keystroke is 8.4 bits, while for English with a 27-key board it is only 4.8 bits. Multiplying these by the mean lengths we obtain surprisingly similar figures of 23 bits/ideographs for Chinese and 28 bits/word for English.

characters), and in many cases only alphabetics and numerals are allowed (ie not "/", "!", etc.). Certain characters sometimes have special meaning in certain contexts. For example, many operating systems allow wild-card specification in filenames, either of a single character (often indicated by "?") or a string of characters (by "*"). Programming languages usually prohibit identifiers beginning with a digit, and some (eg BASIC) restrict their length to two characters.

Active
These symbols must agree exactly with internally-stored keywords. Hence they must be translated by the preprocessor, if operating-system modifications are forsworn.

One serious difficulty arises when text is entered in one mode and subsequently used in another. This happens, for example, when a command file is entered using the editor (as passive text) and is subsequently interpreted by another subsystem which normally accepts input from the keyboard. The keyboard input typically contains a mixture of active, neutral, and passive text (say keywords, filenames, and comments), and whenever a subsystem is being entered the preprocessor must obviously ensure that these elements are converted appropriately. However, when a previously-entered text file is used for input, the conversion cannot be done by the preprocessor because

— when the file was entered originally, the preprocessor could not know that it was destined for use by the particular subsystem as a command file,

and

— when the subsystem reads the command file, its contents do not pass through the preprocessor and so there is no opportunity for conversion.

For a more concrete example, consider interacting with the BASIC language subsystem via an ideographic preprocessor. BASIC keywords are typed as ideographs, and converted by the preprocessor to English — PRINT, IF, GOTO, etc.. Furthermore, BASIC identifiers are also typed as ideographs, and must be converted to BASIC variable names — A0, A1, A2, etc. — but let us ignore this at present. The preprocessor must maintain tables to effect the conversion. Now if the BASIC program is SAVEd, what should the resulting computer file look like? If it is to be reLOADed by the BASIC subsystem, the keywords in the program should be in English form, so that they are recognized by BASIC on reloading. If, however, it is destined to be printed on a lineprinter by a standard system utility, it should have been SAVEd as text, with keywords represented as the passive encoding of the ideographs which were typed originally.

For the preprocessor to work transparently, it must invoke mode-conversion programs automatically whenever the user issues certain commands (like SAVE a BASIC program). The virtue of our semiotic classification into passive, neutral, and active signs is that it allows us to see clearly the need for this.

Coding ideographs

For an ideographic terminal to interact with English computer systems, all ideographs must be converted to English pseudonyms. It is possible to identify several desirable properties that the translation should have.

The requirements for passive text are:

1. An ideograph's English pseudonym should be as short as possible, for the sake of storage economy on the host computer.

2. The translation should be fixed-length, so that every ideograph is represented by the same number of ASCII characters.

Requirement 2 will assist in string decomposition by character offset, for then only a fixed scale factor is needed to translate from an ideographic offset to the equivalent one on the host computer. It also simplifies the problem of document preparation, for a fixed-length representation means that existing software for line splitting and justification will work on ideographic text. However, it rules out the rather attractive proposition of directly encoding the radicals forming a character.

3. Given two strings of ideographs, it should be possible to tell from the translations if one is a substring of the other without false matches being caused by incorrect alignment.

This allows matching by context, as used in most editor programs.

4. True English should never masquerade as translated ideographs.

It will often be necessary to mix English and ideographic text. For example, programmers may wish to use some English variable names, for mathematical symbols like x, y, sin, and cos are used in their Western form in most Chinese mathematics. Clearly, requirement 4 can never be fully satisfied, for any sequence of ASCII characters can be entered directly if desired. However, we wish to minimize the likelihood that the kind of text normally entered will masquerade as translated ideographs. It would seem sensible to include a little-used control character to flag each translated ideograph, but we will see below that this introduces difficulties.

For neutral text, requirement 1 has even more force, for filenames and identifiers are usually heavily restricted in length. Clearly, in the case of BASIC identifiers no universal ideographic encoding will suffice, for only 26 + 26*10 = 286 different combinations are allowed, and this is far less than the number of ideographs. Leaving aside this special case, which must be treated as an exception by the preprocessor, neutral text presents another requirement:

5. Translations of ideographic strings should not violate any restrictions that the host computer places on filenames, and should conform with the syntax of variable names in as many programming languages as possible.

This requirement is rather stringent, because for most systems it rules out all characters except letters and digits. Furthermore, the fact that most languages prohibit identifiers which begin with a digit means that we cannot use the standard 4-digit telecode,* as employed, for example, in some standard character indices.[12] However, we will present a simple coding scheme which can represent 27,000 ideographs as three-character strings, with the characters chosen from the digits and upper- and lower-case alphabetics. This allows two ideographs to serve as a 6-character filename, provided that the host distinguishes upper- and lower-case characters.

Another requirement is

6. Although the translation from ideographs to English may be slow (because ideographic text is entered slowly on the keyboard), its inverse should be fast so that the host's output can be displayed on the terminal quickly.

Requirement 3 implies the use of a delimiter to ensure correct alignment of ideograph boundaries when English translations are matched. This is difficult to accomodate within a three-character translation, since with one reserved for the delimiter insufficient combinations of alphabetics and digits exist. To solve this, more than one character is employed as a delimiter. With N possible characters out of which n are reserved for delimiters, $n(N-n)^2$ different triples exist. Examination of this function, where N has value 62 (26+26+10), shows that the obvious choice of a single delimiter gives a paltry 3,700 combinations compared with the maximum of 35,300 when n = N/3. We choose an easily-recognizable encoding with a digit in one fixed position and alphabetics filling the other two. This gives 27,000 combinations. To minimize the likelihood of mistaking genuine English for ideographic translations, the digit occupies the central position, flanked by alphabetics. This has the additional advantage of satisfying the syntax of variable names for most computer languages (except BASIC, and upper-case-only languages).

Requirement (6) indicates that the code should be based on the address of the picturegram for the ideograph in the preprocessor, so that although translation from the ideograph keystrokes to the pseudonym may be slow (because a table of radicals, which may reside on backing store, must be consulted), reconstruction of an ideograph from its translation requires access to the character generator only.

*A brief account of the fascinating history of telecodes is given in reference 2.

Translating interactive languages

As a prelude to our examination of the translation process for BASIC, the most commonly-used interactive language, let us consider a simpler, although somewhat more esoteric, interpreted language — LISP.[13]

LISP

The fundamental entity in LISP is an atom, and is represented by a string of up to 30 letters and digits. The indivisibility of atoms, the generous maximum character length, and the starkness of the LISP syntax make translating an ideographic version an easy proposition. All names in LISP are atoms, including functions and the built-in functions which comprise the LISP system itself. Atoms are combined into lists using a handful of special characters like parentheses, full stops, and quotation marks. These special characters are all commonly used in Chinese text, and so there is no need to translate them.

An ideographic LISP preprocessor must translate each ideograph into a unique sequence of English characters as described above. Translations of the system-defined atoms, of which there are somewhat over 100, need to be stored in a table in the preprocessor with their English equivalents, so that they are converted into the correct character strings.

With the encoding developed in the previous section, 30-letter atoms permit up to 10 ideographs to be concatenated to make a name, and this should be quite sufficient to allow mnemonic identifiers to be used. The only problems with the scheme are the atom decomposition functions EXPLODE and IMPLODE, which do not appear in "pure" LISP but are often provided in specific implementations to break an atom into its constituent characters and reform it. They must be handled by writing special ideographic EXPLODE and IMPLODE functions, which take into account the structure of the codes and use the primitive EXPLODE and IMPLODE to decompose an atom into 3-character pieces. Digits and operator symbols will need to have their single-character representation, and the new atomic decomposition functions should distinguish genuine ideograph translations from ordinary text, using 3-character decomposition

for the first and single characters for the second.

BASIC

The baroque syntax of the BASIC language gives rise to many more problems than with LISP. There are four syntactic categories in BASIC that may be presented as ideographs: keywords, identifiers, character strings, and comments (REMarks).[14] Other elements of the language, namely numbers, arithmetic operators, and special punctuation symbols such as commas and quotation marks, are used in the Chinese language in the same way as in English. Furthermore, the Chinese use ordinary Western mathematical language, so we do not envisage translating the names of mathematical library functions like sin and cos.

Keywords are stored in a Chinese-English translation table in the preprocessing computer. Single ideographs are used for keywords, and although this imposes a degree of unnaturalness on the Chinese representation, the resulting economy of keystrokes in entering programs was judged to outweigh any artificiality. In fact, multi-ideograph keywords could be accepted equally well if so desired.

Identifiers in BASIC comprise an alphabetic letter which may be followed by a decimal digit. In Chinese, identifiers must comprise a single ideograph. Whenever an ideographic identifier is entered in a BASIC program, it is checked against the translation table. If it does not appear, it is added to the table with a 2-character translation. Thus the first ideograph which is not a keyword will translate to "A0", the second to "A1", and so on. Numerals, operators, and punctuation pass through the processor without translation. So also do English letters: this makes the filter transparent to English BASIC.

If English and ideographics are mixed in a BASIC program, confusion may occur. The user cannot tell what English pseudonyms have been assigned to his ideographs, and so cannot guarantee to avoid variable name clashes. The ambiguity could be removed by translating English identifiers to a name selected by the preprocessor, in the same way that ideographic ones are. An easier possibility is simply to forbid mixed-language programming.

Some English letters appear in Chinese programs — we have already mentioned mathematical functions. It is important to ensure that no parts of legal English strings can masquerade as translated identifiers; this is indeed the case for the 2-character identifiers A0, A1, ..., Z9.

Character strings are the most difficult items to translate, because BASIC contains string-processing functions such as LEN() (length of a string), LEFT$(), RIGHT$(), MID$() (substrings), and INSTR$() (searches one string for the first occurrence of another). It is not feasible to encode a sequence of ideographs as a single unit, for this would prevent decomposition. Instead we translate the ideographs individually into fixed-length English strings. The 3-character encoding outlined above is quite suitable, and has the advantage that the English representation is as short as possible. This is important because otherwise string overflows will occur often within the BASIC interpreter.

The BASIC string-processing operations specify offsets in a string as character counts. Since strings now contain a fixed number of English characters per ideograph, all of these figures must be adjusted to account for the new unit of measurement. Thus, since ideographs are converted into three English characters each, @LEN (the ideograph for LEN) is translated into (1/3)*LEN, @LEFT$(..., <expression>) into LEFT$(..., 3*(<expression>)), and so on*. Identifying <expression>s when translating LEFT$, RIGHT$, and MID$ is the closest the preprocessor gets to the syntax of the BASIC language.

Note that this scheme will not work if English and ideographics are mixed within strings. In a simple system, one might choose to outlaw this. However, since symbols such as punctuation and digits count as English, this requirement may be too stringent. The only alternative, if string decomposition is to work properly, is to pad each English character that appears within a string to the length of the ideograph translations. If the pad character is chosen as a control character which would not otherwise appear in strings, it will be easy to remove when translating character strings received from the host on output; however, since one use of strings in BASIC is as filenames, and the host operating system will probably not welcome control characters in these, it is better to pad with a printing character instead.

String operations which involve ASCII character codes, for example CHR$() which returns the character corresponding to a given ASCII code, and CHANGE() which transfers a string to an array of ASCII codes, are not implemented. The most sensible interpretation would be to return the 4-digit telecode mentioned earlier. This would involve communicating with the ideographic preprocessor, and so would need a non-standard implementation of BASIC on the host.

As for program comments, any ideograph in a BASIC REM statement is converted to an English pseudonym using the same translation as for strings. There is no need to pad English characters, but it may be best to do so for the sake of uniformity.

*The "@" prefix indicates that an ideograph is typed; thus "@LEN" should be read as the ideograph whose meaning is LEN.

Lastly, it should not be forgotten that error messages originating from the host computer will have to be translated before being presented on the terminal. Some BASIC implementations simply return an error number, like "?16", which does not need altering. (Note that our aim is not to enhance the programming language, but to preserve it — warts and all — wherever possible.) If error messages are used, they should appear in full in the translation table — a word-for-word translation would probably be too confusing in most cases. Fortunately, BASIC implementations do not include user-defined variable names or parts of program statements in error messages.

Implementations. With the above considerations in mind, we sketch the working of both a simple preprocessor and a more sophisticated one. The first maintains a single translation table, which is initialized to hold the keywords of BASIC. Every ideograph input is translated via this table, which is augmented if the ideograph is absent with the next unused member of the sequence A0, A1, ..., A9, B0, ... as its English translation. Anything received from the host computer is inverse-translated using the table. Digits, operators, and punctuation pass transparently through the preprocessing filter in both directions. English characters do too, unless they appear as translations of ideographs in the table, in which case they are transformed back to ideographs on output. The only syntactic checking of the BASIC program by the preprocessor is in detecting, bracketting, and halving expressions which form the second argument of a LEFT$(), MID$(), or RIGHT$() function, and the expressions can be detected easily by stacking parentheses.

This simple system will work correctly for all-Chinese BASIC, providing punctuation is avoided in strings which are decomposed. The maximum number of different ideographs which can be used in any one interactive session is 260. It will work for all-English BASIC and Chinese BASIC with English identifiers and strings, provided the string decomposition functions are typed in English. It will work for mixed English and Chinese if English is avoided in decomposable strings, unless name clashes occur. These can be avoided by using single-letter variables and ensuring that numbers are not adjacent with letters in strings.

The more sophisticated preprocessor uses BASIC syntax to distinguish strings and comments from identifiers. A table is maintained for identifiers as described above. English identifiers are translated, as well as Chinese ones, to avoid name clashes in mixed programs. Ideographs in strings and comments are translated to a fixed-length character representation, like the one developed above, which cannot clash with keywords, identifiers, or numbers. English characters, punctuation, and digits, occurring in strings, are padded to the same length. Note that an ideograph may occur both in a string or comment and as an identifier. This causes no special difficulty.

What inadequacies still appear in this second preprocessor? Juxtaposition of characters on output cannot masquerade as ideograph translations because characters in strings are always padded. (The padding is, of course, removed before final output.) If the user types English, punctuation, or Chinese as input to his program it will all be translated before going to the BASIC program on the host. There is only one difficulty. If numbers are typed as input, there is no way that the preprocessor can tell whether they are destined for string input:

```
INPUT A$,
```

when they must be padded so that they can participate sensibly in string comparisons; or for numeric input:

```
INPUT A,
```

when padding would cause a BASIC error. Thus we must rule that numbers must never be read into strings. Punctuation and operators that accompany numbers ("+", "-", "E", ",") must be recognized through context analysis by the preprocessor, and padded only in non-numeric contexts. To remove this restriction would require interpolating a hidden subroutine call whenever a number is read, to transfer it into a character string, process it to remove padding, and return the numeric result. If this were done at BASIC level by the preprocessor problems would occur when listing the program, and highly confusing error messages may appear.

Text processing and database access

The handling of unstructured text is rather easier than programming languages for the preprocessor, because of the uniformity of data representation and the syntactic simplicity of editor commands. Ideographs are translated into the universal byte representation described earlier.

Text editing

The variety of text editors is so great that it is difficult to say anything about ideographic conversion which is generally applicable. A simple editor is discussed in some detail in reference 15. It poses very few problems for ideographic conversion. The preprocessor must distinguish several different kinds of information. Editor commands are translated from Chinese according to a pre-defined table. It is possible to determine whether typed input is a command or not by mimicing the editor and entering text mode on the appropriate commands; but a more robust alternative is to examine the prompt sent by the host system and decide on that basis whether a command or text is expected. Another kind of input is the numbers which follow certain commands. Where these refer to line numbers, they pass directly through the preprocessor. If intra-line editing is by character offset, such offsets must be adjusted to allow for the three-character per

ideograph representation of passive text, but unfortunately this precludes editing of mixed English and Chinese files. Character strings specified in contextual search and change commands are treated as text, the ideograph delimiters ensuring unambiguous matching. Another information type used in text editing is the filename, but this causes no special problems because the coding scheme for ideographs does not use English characters that are unacceptable in filenames.

Document preparation

Text files are frequently used as input to document preparation software. In this case, the text will be interspersed with formatting commands, which are identifiable in some way by the formatter program. (For example, a common convention is to place formatting commands alone on a line beginning with a full stop.) Such commands could be made known to the preprocessor when entering or editing the file and translated accordingly to their English equivalents, but the modularity of the system is enhanced if they are placed in the file as passive text and converted to their active form by a special program on the host just prior to formatting. This can be invoked automatically by the preprocessor whenever the document preparation program is run, in a manner which is transparent to the user.

Little difficulty is foreseen in taking advantage of many of the features of English document preparation systems which may exist on the host. In order that filling will work properly, ideographic translations should be separated by white space, and this can be done by the above-mentioned conversion program. Furthermore, ideograph representations should have a fixed length. Any English which appears in the text should not be padded, or else it will occur in the output with ideograph spacing. Automatic hyphenation should of course be suppressed.

Specification of line lengths, vertical and horizontal spacing, centering, and paragraphing will all be effective. It may be desired to convert numerical specifications of character positions so that ideographs rather than characters are measured. Sections of English text will be formatted with a different paper width than Chinese, for in reality each ideograph occupies less width than its English pseudonym: this can be corrected by the user inserting line length commands before and after the English text. Footnotes, superscripts, and subscripts cause no special difficulty. As with English text, underlining and alternative type fonts are allowable only if the output device and its handler permit.

The output of the document preparation program will be a new version of the input with modified line breaks and some space characters inserted for padding (for example, at the beginning of paragraphs). Conversion of this file to paper form is a job for a specialised Chinese application program with knowledge of the character representations.

Access to database systems

Like editors and document preparation systems, database systems vary so much that generalizations about the feasibility of ideographic translation are impossible. An example system[16] has been considered in some detail,[15] and indicates that no special difficulty should occur.

Interaction with the host operating system

The host operating system itself is treated by the preprocessor simply as another interactive subsystem. The keywords, with their ideographic translations, are tabulated in the preprocessor. Wild-card specification of characters in filenames is easily handled by defining a "wild-ideograph" character and expanding it into three adjacent copies of the host operating system's wild-card character.

A distinction must be made between commands which enter an interactive subsystem and those which do not. The latter class comprises logging in and out and the file maintenance commands, including that which invokes the (non-interactive) document preparation system. Because file names and text files are represented on the host system according to the standard translation method described earlier, no special action need be taken by the preprocessor for most of these commands. In the case of the command which sends a file to the host's printer (as opposed to the one which lists it on the terminal) the appropriate ideograph should translate to the name of a special application program designed for ideographic printing. We saw earlier that execution of the document preparation system should be preceded by passing the file through a software filter which translates ideographic formatting commands into English: this action is performed automatically when the document-preparation command is received by the preprocessor.

Whenever an interactive subsystem is entered, information appropriate to it needs to be installed in the preprocessor. This will certainly include a translation table for the keywords of the subsystem, and for most subsystems certain keywords require special action which is specified as code executable by the preprocessor. Since the amount of code is not great, it can be permanently resident in the preprocessor and accessed symbolically from translation tables.

Translation tables for subsystems can either be stored locally on the preprocessor's disk, or centrally in the host. If many ideographic terminals are attached to a timesharing host, there are advantages in central storage — especially if subsystems are being updated or new ones added. Tables are

sent from host to preprocessor by executing a specially-written host application program with an argument which specifies the host's filename of the appropriate translation table. Whenever a subsystem is left, the preprocessor reverts to its table for operating system commands.

All the subsystems we have encountered use the standard method for ideographic translation throughout, except BASIC which builds a table of identifiers dynamically. For this, the table of identifiers must be saved on the host whenever a program is SAVEd, and loaded into the preprocessor whenever one is LOADed. Local storage cannot be used in case another terminal user wishes to run the same BASIC program. It seems that the most generally useful way to store BASIC programs is in the form of passive text, with identifiers and keywords in their ordinary ideograph encoding. This will involve file conversion prior to each LOAD and SAVE command. Part of the table entry for the BASIC LOAD and SAVE keywords will invoke this mechanism automatically, so that the user need never be aware of the process.

Conclusion

This paper has outlined design considerations for an ideographic preprocessor which acts as a timesharing terminal to a host computer, working in English. To the host, it is just an ordinary terminal. None of the operating system needs modifying in any way. A small number of special utility programs and translation tables must be stored by the host: the programs perform such jobs as file conversion and sending translation tables to the preprocessor. They are invoked by the preprocessor when needed, without explicit requests being made by the user.

Six important subsystems have been examined: the operating system itself, and two interpreted languages, which were analysed in some detail; a text editor, document preparation software, and an interactive database management system. Very few limitations need be placed on the user's actions. The most difficult subsystem to handle is BASIC, with its restrictive syntax and non-contextual string-matching operations.

The chief shortcomings of the system are its commitment to half-duplex operation, and the need to forego any processing of characters as numbers. These are both inherent in the nature of the translation process, and cannot be overcome without modifying the host software. It is felt that they are a small price to pay for the convenience of an easily-implementable, modular, ideographic terminal for accessing English host computers.

Acknowledgements

It is a pleasure to acknowledge the contributions of John Brazier, Rod Cuff, Brian Gaines, and Y.H. Ng through interesting and stimulating conversations on the subject of the ideographic preprocessor. This research was supported by GWITS.

References

1. Chang, S.K., "An interactive system for Chinese character generation and retrieval," IEEE Trans. Systems, Man and Cybernetics, SMC-3, pp 257-265, May 1973.
2. Walker, G.L., Kuno, S., Smith, B.N. and Holt, R.B., "Chinese mathematical text analysis," IEEE Trans. Engineering Writing and Speech, EWS-11, pp 118-128, August 1968.
3. For example, the Mullard 11SSDR. Also, Ideographics Inc., Sunnyvale, California make a special printer intended for ideographic output (IPX Model 1800).
4. For example, Monotype International's Lasercomp 4000.
5. Hsieh, Hwang and Lin, "Analysis of Chinese radicals," J. National Chiao-Tung University, December 1972.
6. Kwok, P.C.K. and Lai, J.P.K., "Dual-mode transmission technique for Chinese ideographs in a teletext system," Electronics Letters, 15, pp 268-269, May 1979.
7. Posa, J.G. and Allan, R., "ISSCC: a gallery of gigantic memories, gigabit logic, and single-chip systems," Electronics, 52, pp 138-151, February 14, 1980.
8. IBM Series/1 4978-1 display station, Keyboard-Katakana feature manual, IBM, 1979.
9. Caldwell, S.H., "The Sinotype — a machine for the composition of Chinese from a keyboard," J. Franklin Institute, pp 471-502, June 1959.
10. Kong, L. and Tu, J., "User manual for Loh's Chinese keyboard," Hung On-To Research Centre for Machine Translation, March 1979.
11. Witten, I.H. and Madams, P.H.C., "The Telephone Enquiry Service: a man-machine system using synthetic speech," International J. Man-Machine Studies, 9, pp 449-464, July 1977.
12. Dougherty, C.Y., Lamb, S.M. and Martin, S.E., Chinese character indexes. University of California Press, 1963.
13. McCarthy, J., Abrahams, P.W., Edwards, D.J., Hart, T.P. and Levin, M.I., LISP 1.5 programmer's manual. M.I.T. Press, 1962.
14. Forsyth, R., The BASIC idea, London: Chapman and Hall, 1978.
15. Witten, I.H. and Ng, Y.H., "An ideographic-language front-end processor," Computer J., in press.
16. Held, G.D., Stonebreaker, M.R. and Wong, E., "INGRES — a relational data base system," Proc. National Computer Conference, 44, pp 409-416, 1975.