HL+: High-Level Languages Plus Multilingual M Databases

by Cui Zhang and Richard F. Walters

Abstract

Neither conventional high-level programming languages nor today's general purpose operating systems adequately support database systems. Furthermore, non-English language databases are difficult to treat either in existing database systems or with current high-level languages because they require operations on multiple foreign-character sets in addition to ASCII. This paper describes a means for overcoming these difficulties by accessing the shared-database features of M from other high-level programming languages.

Extensions to this approach to accommodate processing of mixed multiple foreign character strings also are described, and examples of applications follow. This approach is a major shortcut in developing sophisticated multilingual database features, which are important for international database applications as well as applications of multilingual natural-language processing.

Toward Better Database Management

General purpose computing environments are not optimal for database management. Operating systems are designed to meet general purpose needs, which often conflict with the requirements of database applications. [1] Most high-level programming languages lack fundamental features used in database management, such as shared files and automated collation, and persistent data types are unavailable. [2,3,4]

The shortcomings of both high-level languages and database systems in processing English text are greatly magnified in dealing with non-English databases. High-level languages have failed to address questions of processing non-ASCII or mixed-character string data. Foreign translations of database management systems exist for some packages, but there are no truly multilingual systems. Even the monolingual foreign systems fail to provide essential features such as indexing and collation according to natural-language conventions.

International work groups are studying the standardization of high-level language accommodation of foreign character sets [5] and collation according to foreign language conventions [6], but these efforts have not as yet resulted in extensions to high-level languages or database management system (DBSM) packages.

The 1990 ANSI Standard M language addresses many problems described above. English language database requirements are met through incorporating shared files ("globals") as integral parts of the language. Collating ASCII text strings is implicit through functions such as `$ORDER`. Manipulating text strings is accomplished by a number of language operators (notably the pattern match-operator) and by functions such as `$EXTRACT` and `$PIECE`. Its widespread availability on many hardware and operating-system platforms makes it ideal for distributed database systems in a heterogeneous environment. Recent language extensions further enhance this feature by enabling Open MUMPS Interconnect (OMI) between different vendors' systems running in unique operating-system environments.

Despite these strengths, however, the attractive features of the M language remain largely unknown outside the M user community. Few computer science academic programs include M courses, and, as a result, few graduating computer science majors know M's particular capabilities. Users of many M-based commercial systems often are unaware of the language behind their applications.

There are probably several ways to heighten the external visibility of M. One approach is to define areas in which other programming languages are currently deficient, but in which there is a real and growing international interest. Once finding such areas, we should be able to extend the language to solve the deficiencies quickly without requiring the client to abandon more familiar high-level languages.

This article describes the effort to demonstrate that other high-level languages can access the shared database inherent in M globals, and that, through a series of primitive extensions, this database can solve problems of processing multiple natural languages within the same database.

A UNIX-Based Shared-File System

The most significant feature of M with respect to DBMS is its persistent, shared-data file structure consisting of a sparse, tree-form array. M does not require declarations of arrays; instead, it permits dynamic creation and modification of arrays, which then are stored on disk in a shared file accessible by all users as soon as a node has been created or modified. Since these arrays are explicitly created by normal M syntax, there is no need to open external files in order to retain variables created during an M interactive session.

Multiple files can be maintained in a single shared file by giving each "file" a different identifier. M shared files may be viewed as an abstract data type whose internal structure is transparent to the user. The functionality of this data type is defined by the operations available through M commands and functions. Users are able to ascertain the existence and name of the next descendant subscript, the existence and name of the next sibling subscript, and the presence and value of data at the current node.

Taken together, M commands and functions allow users to create and manipulate shared variables, usually consisting of hierarchical structures closely resembling real-world data. It solves the difficulty encountered in relational models attempting to deal with non-first-normal form data [as in note 7]. As will be shown, this same functionality can be accessed without requiring application programs to be written in M.

There are several steps leading to multilingual database-management functionality. First, an independent M shared-file system was created for high-level languages to access. Next, a set of utilities was created to enable operations on the shared files. The multilingual capability resides in the shared files, and additional utilities allow external users access to these features. These steps are described in the next section.

Interface between M Globals and C

The M shared database (globals) in our laboratory is an independent package written in C and running under UNIX on bitmapped workstations. This package is independent of the M interpreter, and users can access it from processes written either in M or other high-level languages. Communication among different processes takes place via the shared database through message protocols that mimic the M functions. The approach is akin to the client/server conceptual model described by Domingo and others. [8] It offers many features useful to a wide variety of database operations.

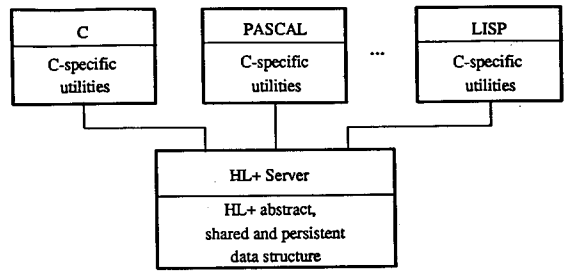
The HL+ concept is accomplished by creating a set of utilities that can be used by high-level languages to perform the operations described above. Using C as an example, we have created the following functions in Table 1 to interface to the independent database system.

Using these functions, it is possible to interface programs written in C to the HL+ database, performing complex tasks on database entities that would be extremely difficult to achieve without this functionality. Automated collation according to the ASCII code set is implicit via the above functions, and variables can be set, accessed, and retrieved for subsequent examination and modification in C. Furthermore, since many high-level languages can call C programs directly, the HL+ functions described earlier could be implicitly extended to those languages by using the C utilities.

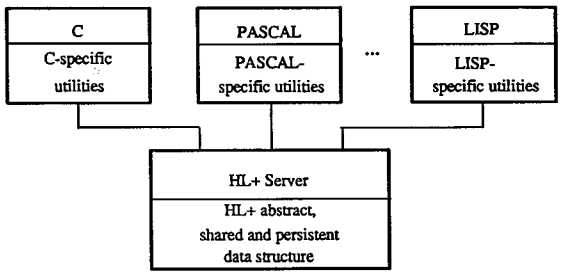
Figure 1 illustrates alternative solutions for accessing MUMPS shared files from multiple HL+ programming languages.

minit()	initializes interaction with the shared database manager.
mhalt()	terminates interaction with the shared database manager.
mset("var(subscr1,subscr2)","datastrng1")	assigns the value of datastrng1 to the node var(subscr1,subscr2)
mkill("var(subscr1)")	kills the node var(subscr1) and its descendants, if any exist.
mdata("var(subscr1,subscr2,. . .)")	returns information about the presence of data and descendants for the node var(subscr1,subscr2,. . .).
mget("var(subscr1,subscr2,. . .)")	returns the data value, if any, for the specified variable node. If none is present, returns a null value.
morder("var(subscr1,subscr2)")	returns the next subscript at level subscr2.
mquery("var(subscr1)")	returns the next complete subscript reference for var, using a depth-first search.
mlock("var(subscr1)")	locks the referenced node and its subscripts for data security.

Table 1. Interface to the independent database system.



(a)



(b)

Figure 1. Alternative solutions for linking MUMPS shared files to other high-level languages. The first (a) illustrates a solution involving HL-specific translation tables for each separate language. The second (b) uses the ability of most languages to call C functions to achieve the same goal.

In this manner, programmers in conventional high-level languages can gain access to the rich set of M database management tools.

Multilingual Extensions to M Globals

As described thus far, the HL+ concept will perform database functions on ASCII character sets, collated according to the ASCII collation sequence. While this capability represents a significant improvement in database functionality, it does not solve problems introduced by applications involving natural-language processing where multiple character sets are used. In a multicharacter-set environment, additional functionality is required. Some key components include creating data strings in non-ASCII (or mixed) character sets, creating subscripts in non-ASCII character sets, and collating subscripts according to algorithms that match the collation conventions of different natural languages.

Each natural language has one or more collation conventions. Each convention is a separate algorithm encompassing a set of collation rules used in dictionaries and other indexed lists in a given natural language. Van Wingen points out that collation algorithms must include the following basic components (applied differently in different natural languages) [9]:

"—expansion:	transforming ligatures into a sequence of single letters
—contraction:	transforming digraphs into a single temporary letter (this involves a certain risk. . .)
—equalization:	transforming capital letters into small letters, or certain accented letters into simple letters
cleaning:	removing irrelevant characters like blanks from the key
mapping:	transforming every single letter (as produced) into a character from the collating sequence of the computer in the right order."

To present a generalizable scheme in HL+ for incorporation of many such collation algorithms, one (or more) for each natural language, we define the following two functions:

HL+ Function	Semantic Action
mnlcs(varname,"charset")	defines charset for subscripts for the named variable
mcollate(varname,"collalgx")	defines collation algorithm for the named variable

These two functions of the HL+ system enable the user to specify any character set to serve as the primary set for subscripts of a named variable, and then to select the desired collation algorithm for ordering those subscripts. The syntax need not be changed for data strings containing non-ASCII characters with these functions (mset, morder, . . .).

HL+, as defined, is rather a polymorphic high-level language that supports database management in a multilingual natural language-processing environment. When the HL+ user specifies a valid pair of character set and collation sequence, a semantic mapping for all DBMS utilities, i.e., a semantic variation of the set of functions specific to the desired character set and collation sequence, results.

Figure 2 shows that the meta-information, describing the natural-language character set and collation, is part of the header information associated with specific global variables.

Consequently, existing programs do not require change for other data sets. The user need not be aware of the semantic mapping, since as they are created, data are added automatically according to the default conventions.

This approach makes it possible to run the same set of application programs on data sets containing different or even mixed-character sets.

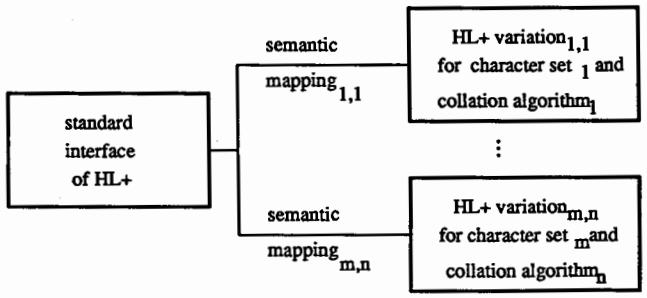


Figure 2. Conceptual model of polymorphic HL+ for multilingual database management. Users can employ the same code to access different databases for multilingual applications.

An additional benefit is the backwards compatibility with extensibility, even when using ASCII characters. A user wishing to retain normal ASCII collation sequences can simply ignore options that specify a different character set or collation sequence.

On the other hand, if a user wishes to achieve "perfect" collation in ASCII, in which uppercase and lowercase letters blend according to the generalized algorithm cited above, all that is needed is a new ASCII collation algorithm.

In multilingual applications, the benefits are even more pronounced. Users can identify default character sets and collation sequences, then enter data in those character sets to achieve desired collation.

In cases where mixed strings involve several character sets (such as directories with both western and Chinese names), multiple collation sets would be required, with a prescribed sequence for displaying the character sets. For instance, in a listing of patients in a hospital where both Western and Chinese names are used, the default character set (say, Chinese) would come first, and Western names would appear in proper alphabetical sequence at the end of the list of Chinese names. Thus two or more collating sets could be used with an appropriate default sequence specified.

HL+ Applications in Multilingual Environments

International epidemiological research and foreign language translation are two fertile areas for HL+ multilingual support.

An international group known as the Universal Medical Information Service (UMIS) needs support for research in patient records from many countries.[10] Machine-readable dictionaries of medical terminology have been collected linking epidemiological terminology in Chinese, Japanese, German, French, and English to the International Classification of Diseases. In a cooperative effort with UMIS, we have shown that by cross-referencing diagnosis information from different countries, the researchers should be able to shed light on diseases with worldwide incidence.

If one investigates a series of national databanks on available records on a specific disease or disease category, a natural way of storing patient cross-index information in each data set would be the following structure:

^ICDAF(icdai,"PT1")=‹pointer to patient file data›
^ICDAF(icdai,"PT2")=‹pointer to patient file data›
^ICDAF(icdai,"PT3")=‹pointer to patient file data›
. . .
^ICDAC(icdai,"PT1")=‹pointer to patient file data›
^ICDAC(icdai,"PT2")=‹pointer to patient file data›
^ICDAC(icdai,"PT3")=‹pointer to patient file data›
. . .
etc.

where ^ICDAF is a variable defined using ISO8859-1 collated affording to French collation rules, and ^ICDAC is based on the Chinese character set GB2312-80, collated according to Pinyin collation rules, etc.

In other words, for each ICDA code there would be a list of all patients with a specific disease whose records were available in a given national database. Each patient number would be separated by a delimiting character, and the patient number thus recorded would point to more detailed records describing the patient's disease, treatment, and outcome. A short program using the utilities described above could be written, requesting the name of each national database set, then retrieving all patients from every national database whose diagnoses matched the desired criteria.

The same process could also be performed in M using this structure:

^ICDAF(icdai)="pptr1 ~ pptr2 ~ pptr3. . ."

where pptr1 is a pointer to patient 1 data, and the concatenated string is manipulated using the $PIECE M function. Although this design is natural and easily processed in M, it may appear arcane to programmers in other high-level languages, who could structure M files to better suit their programming environments. In this way, they would be able to manipulate an M database without having to learn all the details and tricks of the syntax.

Machine-Aided Translations

Our second application is in the area of computer-aided natural-language translation (which some would call foreign language translation). The concept of using translation workstations is new to researchers in the field of machine translation. A translation workstation assists human translators by providing computerized resources such as dictionaries, grammar-writing systems, or expert systems for domain-specific text comprehension. [11]

The HL+ concepts described here greatly facilitate machine-aided translation. Display systems can show independent windows, each with default natural-language character sets. It will be possible to create domain-specific dictionary managers and knowledge-based management systems. Utilities can take advantage of the polymorphic nature of the HL+ system to write at a conceptual level that transcends a specific natural language pair.

Language-independent internal storage is facilitated, so that researchers can concern themselves with building morphologic analyzers and grammar generators that need not be concerned with the specific language pairs used nor the internal storage or representation of each natural language database.

Using dictionary generation as an example, we could create a Chinese-English dictionary in which the Chinese words are collated according to their "Hanyu Pinyin" phonetic representation. The details of internal storage are masked from the programmer, who need only be concerned with Chinese character codes and their English equivalents. The following list illustrates a portion of the code required for this type of system.

```
mnlcs("CEdic","GB2312-80");
mcoll("CEdic","Pinyin");
mset("CEdic([G]/"北/")","north");
mset("CEdic([G]/"矮/")","short; low");
mset("CEdic([G]/"人/")","human beings; person");
etc.
```

Figure 3. Program sections for creating a Chinese-English dictionary.

The identical dictionary definitions can generate a comparable dictionary collated according to Cantonese pronunciation by changing the second line to read

mcoll(Cedic2","Tsang-Chi");

All other lines would remain unchanged, and yet a completely different dictionary would be generated for Chinese readers who prefer the Cantonese dialect.

The details of internal storage are masked from the programmer, who need only be concerned with Chinese character codes and their English equivalents.

We also have begun implementing a comparable collation feature for handling Japanese characters. In Japanese, there are two phonetic alphabets, katakana and hiragana. The former usually is reserved for a large number of borrowed words (many from English), whereas the latter is used to provide grammatic forms required in Japanese as well as spelling words for which no "kanji" (Chinese characters used for many Japanese words) are available. An additional complication is that, unlike a given Chinese dialect, a single Japanese kanji character may be pronounced in more than one way, depending on its word usage. Collation is therefore an unusually complex problem. To date, we have succeeded in defining the mixed-collation sequence used in most Japanese dictionaries today, and we have built an extensive dictionary of Japanese words which ultimately will enable us to collate all such words based on their correct pronunciation.

Extending this concept further, dictionaries for multilingual systems could be created using a natural language-independent internal representation, thereby allowing users to switch source and target languages, concentrating only on the conceptual level of the problem rather than the technical details of internal storage.

Easy Access to M

Database systems today face a challenge of meeting the needs of non-first normal form real-world structures with formal systems comparable to relational model constructs. Soon, these challenges will grow to require increasingly widespread manipulation of multiple character sets. Neither current database packages nor high-level programming languages offer either of these features.

Our research has shown that the complex structures of MUMPS globals can operate independently of the M lan-

guage per se, and that this database can be accessed easily from other high-level languages. The interpreter is not required in existing database systems or in high-level programming languages.

This approach is a major shortcut for developing sophisticated multilingual database features. M's functionality is important for international database applications and artificial intelligence applications. It has proven to be invaluable for applications wherein multilingual natural-language dictionaries are involved, such as multilingual medical research machine-aided translation. [10,13,11]

C Users Take Note

Another benefit of HL+ is that it affords functionality now missing in other high-level languages just when the functionality is desirable but means of embedding it in languages such as C remain obscure. Consequently, it is possible that computer scientists familiar with languages such as C would rely on M globals to accomplish tasks not readily achieved otherwise. In turn, this might increase the visibility of M and ultimately enhance its acceptance.

To demonstrate the utility of this system, we have extended our HL+ database system to include automatic collating options for mixed upper- and lowercase ASCII characters, Chinese Pinyin (for all characters in the GB2312-80 standard), and Japanese mixed katakana-hiragana based on JIS X0208 characters. We also retain the ability to collate on the pure code value of character sets, so that there are currently six different collation algorithms available. We are in the process of implementing collation algorithms based on ISO 8859-1 for several Western European languages, notably French, German, and Spanish. We also have immediate interests in adding Tamil and Korean character sets, with a longer-range goal of extending our efforts into other languages. Eventually, this system will work as a high-level tool in the natural language processing environment based on our multilingual workstation. [12,13]

The research described in this report is the result of an international team effort. Key members include G. Bradfield, E. Clubb, E. de Moel, J. Diamond, J. Domingo, B. Douglass, W. Giere, A. Puig, I. Wakai, and W. Yaksick. The research has been partially supported by General Electric Corp.; the Hok Yindong Foundation; MGlobal, Inc.; the MUMPS Development Laboratory (Nagoya, Japan); Sony Microsystems, Inc.; Sun Microsystems, Inc.; the University of California; and the U.S. Veterans Administration.

Endnotes

1. M. Stonebraker, "Operating System Support for Database Management," *Communications of the ACM*, 24:7 (1981), 412-418.

2. T. Andrews and C. Harris, "Combining Language and Database Advances in Object-Oriented Development Environment," *OOPSLA 87 Proceedings*, 430-440; 1987.

3. M.P. Atkinson and O.P. Buneman, "Types and Persistence in Database Programming Languages," *IACM Computing Surveys*, 19:2 (1987), 105-190.

4. S.R. Ladd, "Persistent Objects in Turbo Pascal," *Dr. Dobb's Journal*, 15:9 (1990), 36-40.

5. ISO/IEC/JTC1/SC22/N776: Japanese Member Body Contribution on Character Handling Requirements in Programming Languages, (working paper for review by ISO SC22 Member bodies), 1990.

6. J. Melton, "Further Character Set Issues (Resolved)," ANSI X3H2-89-376rev2, Dec. 12, 1989.

7. Z.M. Ozsoyoglu and L.-Y.Yuan, "A New Normal Form for Nested Relations," *ACM Transactions on Database Systems*, 12:1 (1987), 111-136.

8. J. Domingo, G. Gradfield, C. Zhang, and R. Walters, "A Conceptual Model for MUMPS Systems," *MUG Quarterly*, 21:2:21-6.

9. J.W. Van Wingen, "Sort Order Schemes in Different Languages," ISO/IEC/JTC 1/SC2 N2111, 1990.

10. D. Walker, "UMIS—Universal Medical Information Service—History and Progress," *Proceedings, Medinfo '89 Conference*, Beijing and Singapore (1989), 790-794.

11. L-C. Tong, "The Engineering of a Translator Workstation," *Computers and Translation*, 2, (1987), 263-273.

12. R.F. Walters, "Design of a Bitmapped Multilingual Workstation," *IEEE Computer*, 23:2 (1990), 33-41.

13. R.F. Walters and C. Zhang, "Support of Multilingual Medical Research," *Artificial Intelligence in Medicine*, 3 (1991), 131-138.

The authors are from the Division of Computer Science, University of California at Davis. Richard F. Walters, Ph.D., is the executive editor of *M Computing*.