DECEMBER 1989 VOL. 12 NO. 4

a quarterly bulletin of the IEEE Computer Society technical committee on

Data Engineering

CONTENTS

SPECIAL ISSUE ON NON-ENGLISH INTERFACES TO DATABASES

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.
IEEE

IEEE COMPUTER SOCIETY

Data Engineering Bulletin is a quarterly publication of the IEEE Computer Society Technical Committee on Data Engineering . Its scope of interest includes: data structures and models, access strategies, access control techniques, database architecture, database machines, intelligent front ends, mass storage for very large databases, distributed database systems and techniques, database software design and implementation, database utilities, database security and related areas.

Contribution to the Bulletin is hereby solicited. News items, letters, technical papers, book reviews, meeting previews, summaries, case studies, etc., should be sent to the Editor. All letters to the Editor will be considered for publication unless accompanied by a request to the contrary. Technical papers are unrefereed.

Membership in the Data Engineering Technical Committee is open to individuals who demonstrate willingness to actively participate in the various activities of the TC. A member of the IEEE Computer Society may join the TC as a full member. A non-member of the Computer Society may join as a participating member, with approval from at least one officer of the TC. Both full members and participating members of the TC are entitled to receive the quarterly bulletin of the TC free of charge, until further notice.

Letter from the Editor

Several months ago, Won Kim asked me to put together the December 1989 issue. Initially, we agreed on graphical interfaces to databases as the topic. As I began looking into the literature and trying to figure what papers to invite, I noticed that there was very little out there that had to do with non-English interfaces to databases. So I set out to find a broad range of papers, all dealing with interface problems that American and European researchers would find unusual.

The individuals who contributed to this issue were asked to put in a lot of effort in a very short amount of time. A few of them had difficulty writing in English. There were also communication problems for the authors located outside the USA. All of the authors deserve a very warm thanks.

There are two papers dealing with Arabic. The first paper, by Elmaghraby, El-Shihaby, and El-Kassas, gives an overview of why Arabic presents unusual problems for the developer of interfaces, and why English-based software and hardware may not be easily adapted to handle Arabic. The second paper represents the Ph.D. thesis work of Ali Morfeq and describes an effort to develop text database management techniques specifically suited for Arabic.

The third paper also deals with textual data. It is by Yaacov Choueka and describes a very aggressive effort at developing a database of Hebrew texts. This effort has produced research contributions that are of use to developers of text retrieval systems in general, not just to developers of Hebrew-based systems.

The fourth paper is by Fang Sheng Liu and Ju Wei Tai, and concerns the development of graphical interfaces. It suggests that a useful way of constructing and categorizing graphical tokens can be based on Chinese character theory. One interesting aspect of this paper is that it is not dedicated to Chinese-based interfaces.

The fifth paper (by Yoshioki Ishii and Hideki Nishimoto) describes the support of Japanese and Korean database interfaces. It gives an overview of why the Japanese and Korean languages present novel problems, and then describes the approach they have taken in developing their system.

I hope that the readers of this issue find these papers to be unusual and interesting.

[illegible]

Roger King
December, 1989

Problems and Peculiarities of Arabic Databases

A. S. Elmaghraby[1], S. El-Shihaby[2], and S. El-Kassas[3]

ABSTRACT

This paper presents a brief overview of Arabic language databases, their problems, solutions, and directions for the future. Experience is based on implementation of an Arabization scheme and several database applications using standard DBMS packages and programming languages. Most of these problems are due to the fact that Arabization is not integrated in the design of the hardware and system programs.

1. INTRODUCTION

The need for Arabic Databases has been recognized since the introduction of computers in the Middle East. Modern electronic computers were first introduced to Egypt in the 1950's starting by the IBM model 1620. Currently all types of computers exist, and are Arabized, with the exception of some high end models.

Arabization schemes for computers are key to understanding problems of Arabic Databases. It is also beneficial to briefly get acquainted with some peculiarities of the Arabic language.

The Arabic language is written from right to left using the Arabic character set with connected script. Twenty eight characters compose the Arabic set, however each character has multiple shapes. Displaying an Arabic word requires knowledge of shape selection of each letter based on the letters positions in the word.

The lack of endogenous computer designs in the Arab world have forced Arabization to follow a modification and patchwork approach. Peripheral Arabization has been the most popular approach in the early years and is used for mainframe and minicomputers while microcomputer Arabization has been approached with a variety of schemes.

This paper presents some of the problems facing Database design and Database interfaces for Arabic applications. Some of the necessary background related to Arabic language and Arabization schemes is also included. One major issue in Arabic language that is briefly presented without detail in this paper is diacritics (al-tashkeel) - which has been dropped in most modern Arabic publications -- diacritics are mainly used to improve pronunciation.

[1] EMACS Dept., University of Louisville, Kentucky, USA 40292.
[2] University of Alexandria, Alexandria, EGYPT.
[3] EB Dept., Eindhoven University of Technology, the Netherlands.

2. PROBLEMS OF ARABIC DATABASES

2.1. Peculiarities of Arabic and Arabization

Differences among natural languages have caused performance degradation in many computer environments, including databases. However, software designers, most of the time, include considerations for the differences in Latin-based languages. They do construct layers to allow the update of the character set -- IBM's MVS OS allows that within the whole OS interface. This is convenient when applications do not use "clever" techniques and communicate directly with system peripherals, such as in the case of Focus on IBM systems.

When the linguistic differences include more than minor variations in the character set, such as in Arabization, serious problems occur. Operating system designers did not anticipate, when designing their supervisor calls handling terminals, that entering a character may alter the shape of an already displayed character. In order to face these problems designers will assist in manoeuvering around the system capabilities to reach these needs.

In an attempt to build an Arabic database, the designer is faced with the obvious problems of using a non-English character set in addition to some peculiarities that relate to Arabic language and Arabization schemes.

2.1.1. Orientation

The Arabic language is written in a right to left orientation. When the first IBM Arabic terminals (XCOM, XCOM1, and XCOM2) were introduced they allowed writing characters in both orientations; left-to-right (R-L), and right-to-left (L-R). However, because of technical problems, not related to Arabic language features, letters entered through these terminals were accepted in the way they appear on the VDU screen. So, in L-R orientation all Arabic strings were stored in an order which is the reverse of their entry order, on the other hand all Latin characters were also reversely ordered if the L-R orientation was used. The Arabic user was forced to switch between L-R and R-L orientations. L-R orientation is essential for database queries and operating system commands and R-L orientation has to be initiated before Arabic strings are entered. The orientation switching also resulted in a screen reversal of strings entered previously in the other orientation. Besides the inconvenience, this concept historically lead to a serious bad practice (also IBM still insists on this mechanism in their new XBASIC terminals). The Arabic user still insists on having selectable orientation rather than being satisfied with one bilingual orientation mechanism. Thus, although other natural languages are also defined on a L-R orientation, Arabic computer users have the added difficulty of considering both.

2.1.2. Bilingual Issues

Perhaps because of colonial influence in almost all Arabic speaking nations, the Arabic user will never renounce the use of at least one Latin language. The Egyptian scientist will use some technical terms in English while the Egyptian artist and humanist will use some

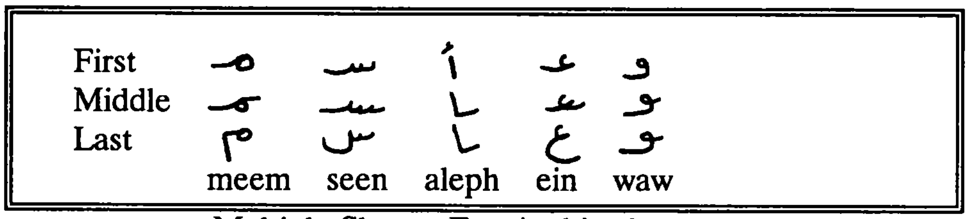
French. This indeed is not very peculiar. Remember the use of French in the Russian novel "War and Peace" of Tolstoy (1869). However when it comes up with two languages each having its orientation, the problems are more serious than adding a few characters from a different character set.

Problems in Database interfaces allowing bilingual entry can be explained by a simple example. Assume upper case letters denote English and lower case letters denote Arabic characters. Then if a bilingual field is entered in the following sequence "ABCxyzDEF", a person reading Arabic will expect it's display as "ABCzyxDEF". If we are interested in sorting this field based on the first four characters, should we view this string as "ABCx" or as "ABCz" ? Storing the data based on it's display format means considering "ABCz" ! Such problems can (and have been) solved using un-shaping operations on such fields before performing any Data base operations such as search or sort. This however introduces extra performance penalties and builds language dependencies deeper in the Database.

2.1.3. Automatic Shape Determination

A character must be displayed in a form which is a function of the context of characters it appears within. Most of the languages prior to typography shared this feature. An "e" at the beginning of a word would be slightly different from an "e" in the middle of a word. However with increased mechanization this feature was relaxed in Latin languages. Writing is considered a fundamental art in the Arabic culture, and is rooted in the predominantly Islamic notions restricting other forms of art such as sculpture and painting. Arabic speaking people have strong feelings for their language and are not expected to give up their art in favor of speeding up the implementation of new technologies.

Automatic shape determination is needed and can be delegated to the computer, or achieved by using an Arabic typewriter style keyboard with an extended character set. Essentially this feature leads to problems in storing even purely Arabic fields. If an Arabic field is entered as "abc" it should be displayed as "cba" and "a" should be displayed using its beginning of the word shape, "b" uses it's middle of the word shape, and "c" uses end of word shape. The sequence "aaa" itself will be displayed using three different shapes for "a". If we were to search for occurrences of a particular character we should search for all three of its representations. One surprising simplification in Arabic that contrasts these complications is the lack of capital letters. Figure 1 displays a sample of Arabic letters and their multiple display shapes based on their position.


First
Middle
Last
meem
seen
aleph
ein
waw

Multiple Shapes For Arabic characters
FIGURE 1

2.1.4. Connected Script

Similar to most languages, hand writing is almost always done using connected scripts. For the same reasons that Arabic printed characters have not been modified to discard automatic shape determination they also have been required to support connected script. This is essentially a challenging problem in some cases. Graphic adapters such EGA and VGA for personal computers do not allow completely connected text in their text modes.

2.1.5. Diacritics

Diacritics in Arabic language present a major problem. They are not really characters, they are character modifiers. If Arabic implementations would consider character-diacritic pairs as the basis of an expanded character set, this would lead to an extremely large character set. The possibilities of an Arabic implementation using more than 256 characters scared most of the Arabic language implementers (although other languages such as Kanji considered this problem). In a more conceptual view the expanded set solution was not view appropriate because it does not reflect the nature of Arabic diacritics. A database query searching for a character in Arabic should result in all occurrences of the letter regardless of its diacritic modifiers. An alternative approach to diacritics which seems logical is to consider the symbols as normal Arabic characters that are not stated in the alphabet. This is consistent with the Latin view of diacritics as vowels. A true discussion and analysis of this problem is beyond the scope of the paper and requires further research.

2.1.6. Numbers and Decimal Notation

The above problems are mainly concerned with textual data. Numerical data provides other types of problems. Some Arabization mechanisms have numeral computational while others do not. Non-computational numerals are needed for manipulation of data. Furthermore modern Arabic numerals are of Hindi origin and their positional value increases left to right in the same manner as English numerals which themselves are the old Arabic numerals. A numeral is considered computational if it can be entered in response to a numeric data entry command. Computational numerals are standard in seven bit ASCII Arabization, while they are not necessarily standard in eight bit ones (Sabri 1986).

Some Arab countries such as Tunisia have switched back to the original (currently Latin) Arabic numerals. In both cases when entering the most significant digit in an Arabic context it's position would follow other characters and entering additional numerals will require left shift of previously entered digits. In contrast, when entering Arabic digits in a Latin context they are entered in sequence and in their appropriate visual position.

The differences in the Continental Decimal Notation (CDN) between Latin languages are minor. In the US a period and a comma are used. A number represented in the US as "1,001,000,555.20" must be written in French as "1.001.555,20". The Arabic culture gave up its decimal notation and at some time started using a Persian notation with different symbols for the decimal point and the thousands delimiters.

3. ARABIZATION SCHEMES

3.1. Peripheral Arabization

Perhaps the most logical scheme for Arabization is based on peripheral Arabization, however;

i. Most of the databases ignore the mechanisms of the peripheral Arabization, so either they tend to inhibit non standard sets to be sent to the terminal, or they fail to trigger the right sequence required by the added Arabization functions. So databases (and other application programs) must be revised for these terminals.

ii. Changes and updates are less convenient with terminals. A change must support earlier products using emulation if compatibility is not provided.

iii. Text handling in graphic modes is generally provided by the application. In database software this is typical in the statistical layers responsible for charts and plots. Even when the operating system kernel supports such processing, most applications find these OS supervisor calls at least non-efficient if not non-sufficient. With the increased interest in Hypertext the issue of coexisting graphics and text is of increased importance.

In general an Arabized peripheral should handle the following functions:

i. Automatic Shape Determination (ASD).
ii. Orientation in L-R and R-L modes.
iii. Representation of numerals.

Some database experience with FOCUS on IBM mainframes using peripheral Arabization of the IBM XBASIC terminal demonstrates bad behavior. Arabic strings are stored in reverse in some modules and are not displayed from others. In addition special characters are incorrectly interpreted since they posses EBCDIC values that do not correspond to their Latin characters. And, as far as numbers are concerned, the modern Arabic CDN is not supported.

The ARABIC VT320 terminal from Digital connected to a VAX functions appropriately using Digital's RMS (Record Management System). However, output on that terminal is inconsistent with output on the CI600Q VAX printer which is Arabized by the same company. Inconsistency can be attributed to some technical variations and some release differences.

3.2. Hardware and Software

Since it is not always possible to invent an Arabized peripheral that will be respected by the DBMS, it is sometimes more practical to create software solutions. Given the DBMS behavior as a finite set of algorithms, it is possible to create software Solutions (patches). Examples of such solutions are patches for some of the popular PC software packages such as DBase III. Also, a similar patch for Turbo Pascal allowed writing B+ tree code that allowed implementation of Arabic databases.

A simplified version explaining software implementation for Automatic Shaping on an Arabic string (no lam alefs & kas chars) is given in Figure 2. This algorithm is presented in Turbo Pascal syntax. Variables of the type CharTable contain an entry for each Arabic character, and each entry contains the different shapes for that character.

```
procedure ShapeArabicString( var Table : CharTable; var Atext : string );
const
  ALONE = 0; LAST = 1; FIRST = 2; MID = 3;
var
  Len,  Pos, I : byte;
  Next, Prev   : boolean;  {are TRUE if the next/prev chars are Arabic}
begin
  Prev := FALSE; Pos := ALONE; Len := Length(Atext);
  for I := 1 to Len do
  begin
    Next := (I < Len) and
            IsArabicLetter( Atext[ Succ(I) ] ) and
            Not IsNumericChar( Atext[Succ(I)] );
    Pos  := ( Ord( Next ) Shl 1 ) + Ord( Prev );
    Prev := Not IsEndArabicLetter( Atext[I] );

    if IsArabicLetter( Atext[I] ) then
      Atext[I] := Table[ Atext[I] ][ Pos ];
  end;
```

Figure 2
Simplified Algorithm For Automatic shape Determination

A layered architecture for Arabization can be used for Database management systems and other application software. Figure 3 depicts a possible structure for this architecture.

For example an application's request to handle the display of an Arabic string can be view as a procedural call to the Arabic Interface. This is given as follows:

```
APPLICATION                ARABIC INTERFACE

Display(Str, SCREEN) --> Shape(Str)
                         Orient(Str, ScreenOrientation)
```

Trans(Str, ScreenCharSet)
ScreenDisp(Str)

```
        Application (DB, ..etc)
                   |
                   | Unshaped text & destination
                   |
           Arabic|Interface
     +-----------+-----------------+
     |         Editing             |
     +-----------------------------+
     |   Shaping / Orientation     |
     +-----------------------------+
     | Device specific translation |
     +-----------------------------+
     |     Device interfaces       |
     +-----------+-----------------+
                 |
                 +----------------------+
                 |                      |
                 |
     +-----------+-------------+
     | low level dev interface |
     |eg char gen, Keyboard def|
     +-------------------------+     .........
     |        Device 1         |
     +-------------------------+
```

Figure 3
Layered Architecture For Arabization

3.3. Standards and Portability Issues

3.3.1. Approaches to Standard Arabic Codes

Arabic Language Peculiarities are compounded by the lack, or more accurately, the multiplicity of standards. The problems of Standardized Arabic codes are due to the absence of a strategic plan for Arabization of information representation in general.

Two different approaches for defining standardized Arabic codes can be identified:

i. An academic approach was initiated by proposing a definition scheme similar to the Latin ASCII and EBCDIC. The problems of this approach are associated with the mapping approximately 30 characters in place of 26*2 characters representing upper and lower case English letters - note that the Arabic language does not have capital letters. The mapping problem occurs when the codes replace characters that are needed by the application software (including DBMS systems). The ASMO 449 and the ARCII standard codes replace almost all the graphic characters. Most database systems use these graphic characters to create their windows!

ii. A commercial approach was based on inserting Arabic codes in place of elements of standard codes thought to be not widely used by application software. This leads to a code page which is aesthetically very unsatisfactory and logically peculiar. Arabic codes in such schemes are sparsed in some code page. The number of standards defined according to this method is almost equal to the number of Arabization layers on PC's. Microsoft introduced an Arabic shell which itself contains four different code pages. One of Microsoft's pages is ASMO 449 compatible, others are called transparent--indicating that applications should run without change, a goal that is not yet fully achieved. In the PC applications APTEC Arabization is a popular standard.

Figure 4 depicts two seven bit standards and samples from the eight bit APTEC which is a de-facto standard.

3.3.2. Portability

3.3.2.1. Differences between Code-Pages

The main differences between one code-page (C-P) and another is mainly in one or more of the following:

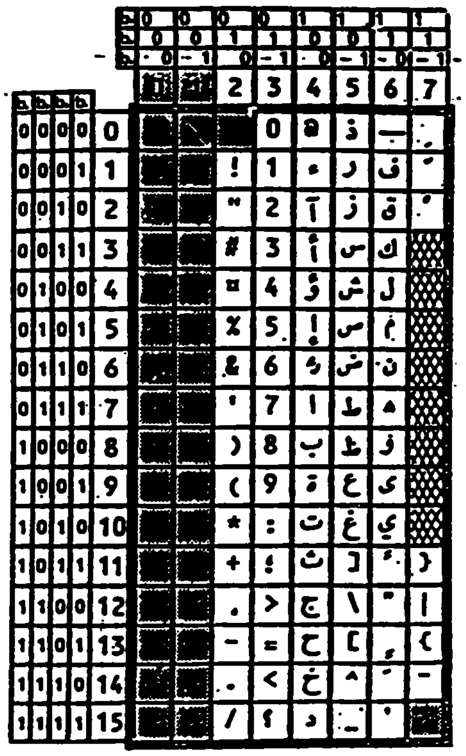
i. Codes of the Arabic characters.
ii. Order of Arabic characters, for example some C-P considers the letter "waw hamza" to follow "alef" while others consider it to follow "waw".
iii. The initial definition of the Arabic character set; some consider "lam-alef" as on character while others treat it as two collating ones.
iv. The set of defined diacritics.
v. The definition of digits and special characters; some C-P redefines codes for numerals and others use a single set for both Arabic and Latin.

Arabic code-pages are either seven or eight bit pages. Seven bit C-P has Arabic letters defined in the same code area of the Latin character code page. It might seem that this allows most Latin oriented applications to work correctly without major modifications, however:

i. Since the number of Arabic characters is slightly greater than the Latin set. Both upper and lower case codes will be needed. Thus applications that make case translations and those that are case insensitive will create problems. Also, the lack of capital letters in Arabic as mentioned earlier is substituted by the need for ASD.

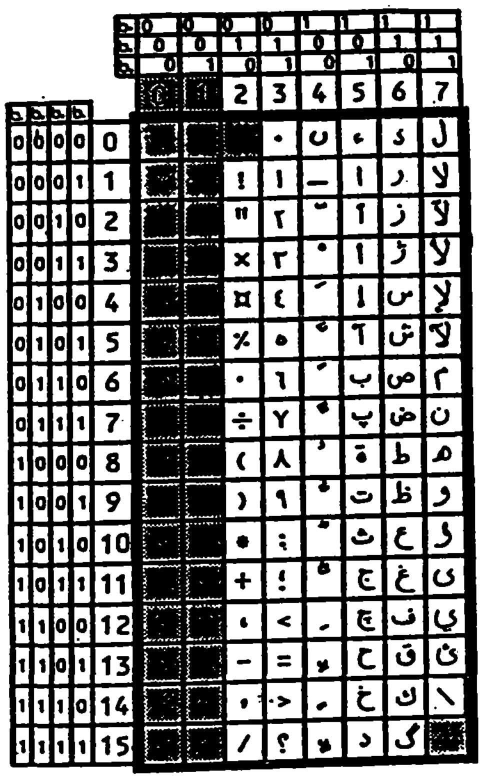
ii. Bilinguality is an ill-defined problem in 7-bit codes. Insertion of control sequences to handle the switching from one code area to another is mainly used and indicates a change in language.

a) ASMO 449



b4	b3	b2	b1	b7 b6 b5 →	000	001	010	011	100	101	110	111
					0	1	2	3	4	5	6	7
0	0	0	0	0	■	■	■	0	@	ذ	ـ	ِ
0	0	0	1	1	■	■	!	1	ء	ر	ف	ّ
0	0	1	0	2	■	■	"	2	آ	ز	ق	ْ
0	0	1	1	3	■	■	#	3	أ	س	ك	▨
0	1	0	0	4	■	■	¤	4	ؤ	ش	ل	▨
0	1	0	1	5	■	■	%	5	إ	ص	م	▨
0	1	1	0	6	■	■	&	6	ئ	ض	ن	▨
0	1	1	1	7	■	■	'	7	ا	ط	ه	▨
1	0	0	0	8	■	■	)	8	ب	ظ	و	▨
1	0	0	1	9	■	■	(	9	ة	ع	ى	▨
1	0	1	0	10	■	■	*	:	ت	غ	ي	▨
1	0	1	1	11	■	■	+	؛	ث	]	ً	}
1	1	0	0	12	■	■	،	>	ج	\	ٌ	|
1	1	0	1	13	■	■	-	=	ح	[	ٍ	{
1	1	1	0	14	■	■	.	<	خ	^	َ	‾
1	1	1	1	15	■	■	/	؟	د	_	ُ	■

b) ARCII



b4	b3	b2	b1	b7 b6 b5 →	000	001	010	011	100	101	110	111
					0	1	2	3	4	5	6	7
0	0	0	0	0	■	■	■	٠	ں	ء	ذ	ل
0	0	0	1	1	■	■	!	١	ـ	ا	ر	لا
0	0	1	0	2	■	■	"	٢	ً	أ	ز	لأ
0	0	1	1	3	■	■	×	٣	ٌ	ا	ژ	لا
0	1	0	0	4	■	■	¤	٤	َ	إ	س	لإ
0	1	0	1	5	■	■	%	٥	ء	آ	ش	لآ
0	1	1	0	6	■	■	·	٦	ُ	ب	ص	م
0	1	1	1	7	■	■	÷	٧	ِ	پ	ض	ن
1	0	0	0	8	■	■	(	٨	ّ	ة	ط	ه
1	0	0	1	9	■	■	)	٩	ْ	ت	ظ	و
1	0	1	0	10	■	■	*	:	ٍ	ث	ع	ؤ
1	0	1	1	11	■	■	+	؛	ه	ج	غ	ى
1	1	0	0	12	■	■	،	<	ـ	چ	ف	ي
1	1	0	1	13	■	■	-	=	ﹾ	ح	ق	ئ
1	1	1	0	14	■	■	.	>	،	خ	ك	\
1	1	1	1	15	■	■	/	؟	ﹾ	د	گ	■

c) Sample 8 bit code

ENGLISH		ARABIC	
LETTER	ASCII	LETTER	ASCII
A	65	أ	193
B	66	ب	194
C	67	ت	195
...		...	

Figure 4
Standards For Arabic Character Set

Eight bit codes are more popular and map the Arabic characters to the upper half of the ASCII set or basically to an unused (in the implementers view) area of the character set. The main problem of such methods is that most applications inhibit codes used in these areas.

3.3.2.2. FILTERS

When text files are stored using ASCII or DIF standards the translation from one code page to another is simple and is performed by creating a software filter. The purpose of a software filter is best explained as a mapping function from one domain to another. Such a mapping function is not a one-to-one function, but typically many-to-many. Some of these filters are sold by software companies to promote the use of their products. An experience in developing such a filter was encountered in an attempt to produce laser output using an HP printer and font cartridge from a database that stored data using a commercial Arabization set (APTEC). When files involve non-text standards such as lotus worksheets the design of such filters is more complex. Also, in some cases where multiple differences exist some manual editing is required after the filtering process.

Some positive experience was encountered in uploading APTEC data from a PC database to an IBM mainframe using a filter. A reverse situation involving transfer of Arabic EBCDIC data to a PC using BABY-36 software posed serious problems.

4. CONCLUSIONS

Problems in Arabic databases and interfaces are mainly attributed to a historically inconsistent approach to Arabization. The basic character set is not yet defined, some definitions include more characters or diacritics. In addition internal representation of each Arabic letter might be based on its display shape or not. This depends on the specific Arabization scheme used. Also, it is often desirable to display bilingual text. This might have Arabic as the base language and English as the "occasional" language or vice-versa.

Experiences in Arabizing DBMS packages and in creating databases using standard programming languages are frustrating. Most solutions are engineering solutions lacking the needed involvement in the design of hardware and system software.

BIBLIOGRAPHY

APTEC, Inc. "APTEC users Manual".

Carrabis, Joseph-David, "Dbase III plus: The Complete Reference" Osborne McGraw Hill, 1987.

Information Builders, inc. "FOCUS Users Manual", Release 5.5, New York, NY. 1987.

Sabri, Ibrahim "Computer Peripheral Arabization", presented in NCR Seminar series 1986, Cairo, Egypt.

Bayan: A Text Database Management System which Supports a Full Representation of the Arabic Language

Ali Morfeq
Roger King

Department of Computer Science
University of Colorado
Boulder, Colorado 80309

Aqil Azmi

Center for Hadith Analysis
Boulder, Colorado 80303

ABSTRACT

It is difficult to use existing computer systems, based on the Roman alphabet, for languages that are quite different from those that use the Roman alphabet. Many projects have attempted to use conventional systems for Arabic, but because of Arabic's many differences with English, these projects have met with limited success. In the Bayan project, the approach has been different. Instead of simply trying to adopt an environment to Arabic, the properties of the Arabic language were the starting point and everything was designed to meet the needs of Arabic, thus avoiding the shortcomings of other Arabic projects. A new representation of the Arabic characters was designed; a complete Arabic keyboard layout was created; and a window-based Arabic user interface was also designed. A graphical based windowing system was used to properly draw Arabic characters. Bayan is based on an object-oriented approach which helps the extensibility of the system for future use. Furthermore, linguistic algorithms (for word generation and morphological decomposition of words) were designed, leading to the formalization of rules of Arabic language writing and sentence construction.

1. Introduction

Arabic language processing has been hampered because, for the most part, people have simply attempted to adopt English-based computer systems for Arabic such as [TAYL86]. This is very difficult and, to some extent, futile because the nature of the Arabic language is much different than that of English. Arabic, for example, is written from right to left; thus, the Arabic sentence structure and grammar are very different from that in English. Therefore, existing systems ("adopted English systems") cannot adequately process Arabic language data. In Bayan, on the other hand, the Arabic properties are served first. This new environment will try to examine the problems in working with the Arabic language by using Arabic text databases as an application of that environment.

This research will address those problems that have led to the shortcomings in the current Arabic programs. In the next section, an overview of the properties of Arabic will be given so the reader may appreciate some of the possible problems in dealing with Arabic using an English-based system. Some aspects of Arabic wordprocessing requirements can be found in [BECK87] [AHME86].

2. The Arabic Language

Arabic is the official language of 22 countries stretching from eastern Asia to northwest Africa. Furthermore, there are other languages which use a modified Arabic alphabet (such as Persian and Urdu). The importance of Arabic is perhaps obvious to the reader.

2.1. Properties of Arabic script

The following subsections describe the general properties of the Arabic script, including characters, digits and writing methods.

Arabic characters

There are 28 characters in the Arabic alphabet. The sounds of 12 of them do not map directly to sounds that are part of the English language. The following are some important properties of the alphabet.

- Arabic may only be written in cursive script.
- Arabic is written from right to left.
- In writing, each character is connected to the preceding or following characters, except six characters, called "stubborn characters". These six stubborn characters are (و ز ر ذ د ا)
- Each character, in general, takes on a different shape depending on whether it appears in the beginning, middle or end of a word or if it is standing by itself (for example, after a "stubborn character" at the end of a word).
- The Arabic alphabet contains no vowels as such. Instead, diacritical marks, called "tashkeel", are used to represent vowel sounds. They are written above or below the consonants. There are 13 "tashkeel". Four "tashkeel" are basic and nine are derived by combining one or more "tashkeel". The "Tashkeel" are essential to understanding the sentence properly; without the tashkeel, a sentence may easily be misunderstood. This is because the placement of the "tashkeel" is determined by the rules of grammar; that is, the "tashkeel" tells the reader if the word is a verb, subject, adjective and so on.
- Each character in Arabic could be pronounced in 13 different ways, depending on the "tashkeel" written with it.

Arabic numerals

Although the numerals of the English language are referred to as Arabic numerals, they differ completely from the numerals used throughout the Arabic world.

There are ten digits in Arabic. These digits are (٩ ٨ ٧ ٦ ٥ ٤ ٣ ٢ ١ ٠). The numbers are written and read from left to right (not right to left, like Arabic words). This implies, obviously, that the least significant digit in a number is the right most digit.

2.2. Existing standards

There have been many attempts to simply modify English computer systems to accommodate Arabic. Such attempts have ranged from modifying input and output (I/O) device drivers to using graphics to draw the Arabic script within application programs. All of these projects have failed in their attempt to represent all of the aspects of the Arabic language. For example, many of them do not have any means of representing the "tashkeel".

One of the proposed standards for representing Arabic characters in computers may be found in [ALSA86]. That work contains a proposed representation of a 7-bit Arabic character set (without Tashkeel) and an 8-bit set for both Arabic (without Tashkeel) and English Still these sets are not able to meet all the requirements of satisfactorily representing the Arabic language. The source of these shortcomings, again, is because these character sets were intended to work with English machines.

In general, the proposed standards lack the following:

- The ability to represent all of the Arabic "tashkeel", especially the ones that are used as a combination of "tashkeel" (e.g., " سُّ ").

- In general, they represent "tashkeel" as characters added to a word. Two words may have the same characters but differ in meaning because of their different "tashkeel". When considering "tashkeel" as characters added to a word, this difference cannot be taken into consideration.
- They arrange some of the characters with special markings as unique characters while they are simply the same character with different signs. For example, { ؤ ، ئـ } are all the same character, called "hamza". The system must consider all of them as one character and not as different characters.
- Finally, these standards are not accepted by all Arab countries and, therefore, until now there is no uniformity on this matter.

Most of the proposed standards are simply derivations from [ALSA86]. Some simply change the ordering of the characters while others remove or add some extra characters. There are some application programs which which tried to solve some of the problems related to the "tashkeel" by creating a character set which included each of the "tashkeel". However, since there are only 127 character entries in the 7-bit extended character sets, they were forced to ignore some of the "tashkeel" in their implementation.

3. The Bayan environment

The Bayan environment attempts to solve the problems described above by attempting to fully represent Arabic script. In particular, representing the different forms of "tashkeel" is given high priority. Bayan also includes a database of Arabic words, roots and derivations rules. This environment could be a base for future Arabic natural language processing research.

The Arabic character set

A 16-bit coded method is to be used in the Bayan system. Arabic characters are encoded in such a way that each character is represented by one and only one code. That is, the "tashkeel" do not affect the value of the character. Instead, the "tashkeel" are treated as a description or a property of the character. They can be treated in the same way underlining, highlighting and other screen display information are treated. A character will be represented by a byte while attributes (such as underlining, highlighting and so on) are represented by another byte. It was decided to represent Arabic characters as in Figure 1. In this figure, the right hand side byte is used to represent the characters while the left hand side byte is used to describe the properties of the character (i.e. the "tashkeel").

attributes "HARAKAT"	Arabic character code

Figure 1
A proposed representation
of an Arabic Character

It may be possible to make the code-byte values a modification of the proposed standard found in [ALSA86] in order to allow some compatibility with the proposed standard. However, the character forms that represent the "tashkeel" are removed as, it is argued here, it is more efficient not to include the "tashkeel" in the character set; rather,they should be considered as attributes to each character. Furthermore, it is also proposed here, that each character should be stored in one code only.

Output drivers must be able to determine the proper shape of each character (depending on where it occurs in the word) along with the correct positioning of the "tashkeel". This is done by building a contextual analyzer which knows how Arabic script is written.

An Arabic keyboard

Since there are only 28 characters in the Arabic alphabet, it is possible to use an English keyboard and simply modify it to resemble the Arabic typewriter keyboard. The keyboard Bayan is shown in Figure 2. Note that the "tashkeel" have keys on the Bayan keyboard, but Arabic typewriters typically do not have "tashkeel" (the tashkeel are added by hand). Device drivers should be able to map the scanning code from the keyboard to the appropriate Arabic code and attribute. This keyboard layout is a collection of different proposed layouts and it overlaps with many of them.

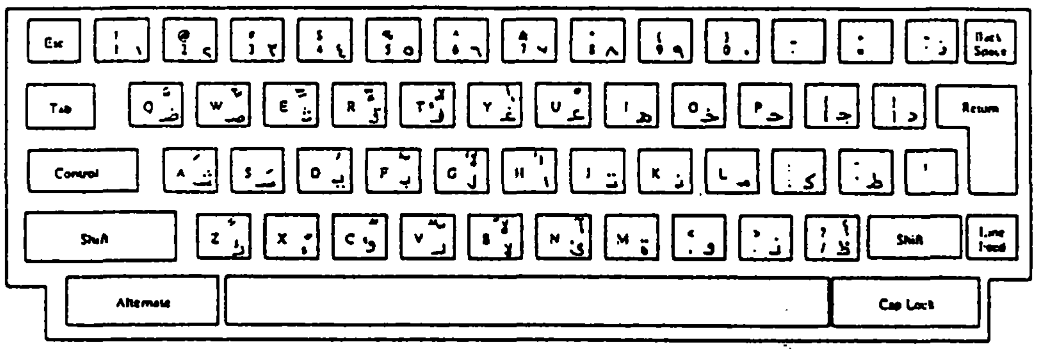


Figure 2
Bayan's Keyboard Layout.

The default input method is to enter the Arabic character followed by its "tashkeel". However, an editor has been added in order to allow the user to first type characters without entering the "tashkeel" and to go back and add the proper "tashkeel" later if so desired. This feature was added because some users prefer to write a word first and then go back and add any desired "tashkeel".

4. Contextual analysis

In Bayan, a contextual analyzer was built. The role of this analyzer is to determine the shape of the Arabic character according to its position in a word. The analyzer takes into consideration whether the character appears at the beginning, middle or end of the word. It will represent the character properly and also place the "tashkeel" in their proper place.

The block diagram of the contextual analyzer is shown in Figure 3. It uses four fonts for the output of Arabic characters. There is a separate font for each possible position or shape of a character. These fonts are : "stand alone font", "start of word font", "connect at middle font", and "connect at end font". The analyzer will take the current and previous character codes and their shapes as its input and will return the correct shapes of the previous and current character. This means that the calling procedure to the contextual analyzer should check if the previous character changes shape after the return from the contextual analyzer call.

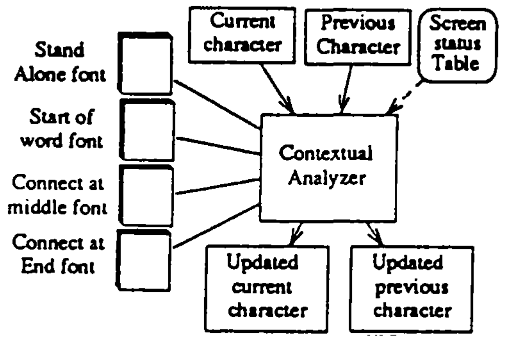


Figure 3
Bayan's Contextual analyzer

The algorithm to determine the shape of the character at the current position is shown in Algorithm 1. It uses the following structure and identifiers.

```
Data structures and constants.
   identifier:
        START_POSITION, MID_CONNECT, END_CONNECT, and STAND_ALONE;


   Arabic_Char: Structure of {
      Code : Byte;       /*Arabic character code*/
      Tashkeel: Byte;    /*Tashkeel code*/
      Shape : int ;      /*character shape  number*/
     }
/***********************************************************************
   Conextual analysis for the Arabic language
   written in C like language. Stuff between '/*' and ' ' are comments.
*/
Procedure ( current_chr: Arabic_char, prev_chr:Arabic_char)
{   current_chr.shape = MID_CONNECT ;          /* default shape */
   if( CAN_NOT_CONNECT( prev_chr.code)){       /* check the prev_chr */
      current_chr.shape = START_POSITION;
      exit();
   } /*   if can_not_ connect */


   if( NON_LETTER(current_chr.code)){      /*check current character*/
      Current_chr.Shape =  START_POSITION; /* current will be start_shape*/
      do case ( prev_chr.shape){
          case START_POSITION: prev_chr.shape = STAND_ALONE;
                    break;
          case MID_CONNECT:   prev_chr.Shape = END_CONNECT;
                    break;
          case END_CONNECT:
          case STAND_ALONE:   break;
          default :           ERROR("SHAPE code");
      } /*do case */
  } /*  if non_letter */
  else {
      do case (prev_chr.shape){
          case START_POSITION:
          case MID_CONNECT:   current_chr.shape = MID_CONNECT;
                    break;
          case END_CONNECT:
          case STAND_ALONE:   current_chr.shape = MID_CONNECT;
                    current_chr.Shape=MID_CONNECT;
                    break;
          default:            ERROR("ERROR in shape code");
      } /* do case prev_chr.shape */
   } /* else */
   return();
} /* end of procedure */
```

Algorithm 1
Arabic contextual analyzer

The identifiers above are used to represent the different fonts used in our system. For example, START_POSITION is the font that contains the shape of all characters when they are at the starting position of a word.

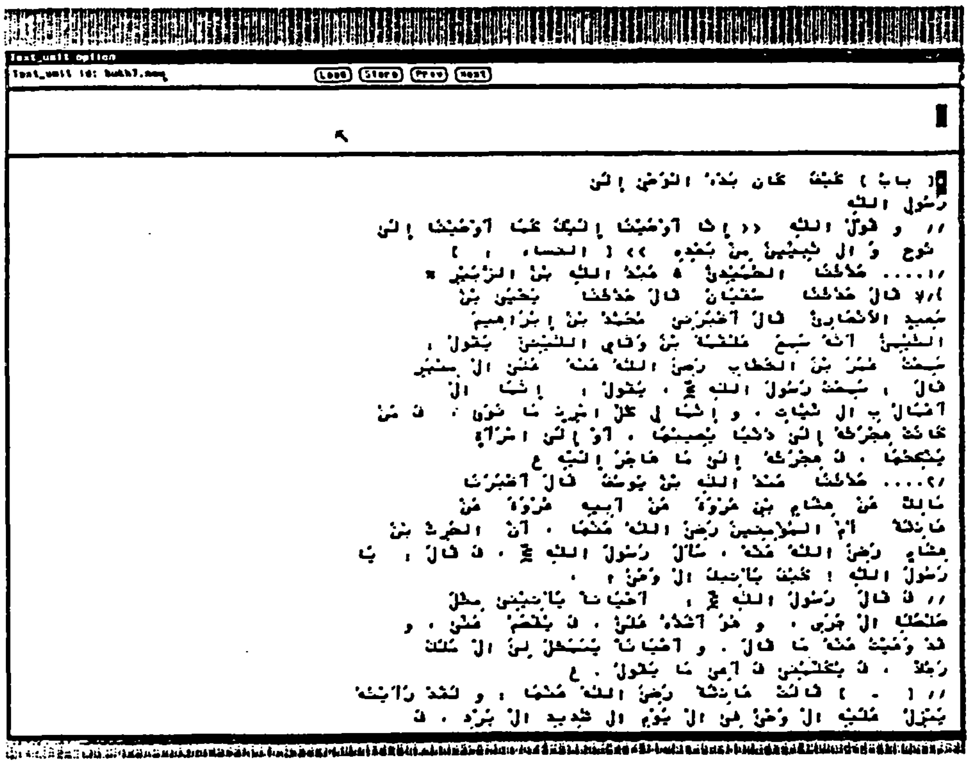
The contextual algorithm will take the current and previous characters as inputs. The contextual analysis alogrithm will examine the previous and current characters. If the previous character can not connect to the current character (e.g. space, comma, a stubborn character,...), then it will make the current character's shape to be 'START_POSITION_SHAPE' and it will exit. Next, the algorithm will check if the current character is a space, a puctuation mark, or a digit; in which case, it will set the shape of the previous character to be the proper ending shape depending on the shape of its previous character(i.e. the second to the last character). If both of the above tests fail, the algorithm will change the current character to connect to the previous character depending on the shape of its previous character.

In the following table, we show an example of how contextual analysis determines the shape of the Arabic character displayed. The first column shows the current input character. The second column shows what is being displayed as a result of the input so far. The third column contains comments of what has been done. The black rectangle is the cursor.

Input character	Output on screen	Comments
س	س▮	start of word shape.
ل	سل▮	connect the second to the first letter.
ا	سلا▮	a ligature is a must, so go back and join both into one new char.
م	سلا مـ▮	start of word shape because the previous can not connect to next.
	سلام ▮	space cause the last letter to have ending shape.

5. Bayan's user interface

Bayan's user interface was built in a SUN workstation environment. Machine dependency was kept to a minimum. The existing interface will allow the use of Arabic text in windows; it will also perform all proper input and output operations on that window. Furthermore, Arabic text may be edited in these windows.



(باب) كيف كان بدء الوحي إلى
رسول الله
،، و قول الله << إنا أوحينا إليك كما أوحينا إلى
نوح و ال نبيين من بعده >> (النساء ،)
،..... حدثنا الحميدي ‌ عبد الله بن الزبير *
)،٧ قال حدثنا سفيان قال حدثنا يحيى بن
سعيد الأنصاري قال أخبرني محمد بن إبراهيم
التيمي أنه سمع علقمة بن وقاص الليثي يقول ،
سمعت عمر بن الخطاب رضي الله عنه على ال منبر
قال ، سمعت رسول الله ﷺ ، يقول ، إنما ال
أعمال ب ال نيات ، و إنما ل كل امرئ ما نوى ، ف من
كانت هجرته إلى دنيا يصيبها ، أو إلى امرأة
ينكحها ، ف هجرته إلى ما هاجر إليه ع
،٢.... حدثنا عبد الله بن يوسف قال أخبرنا
مالك عن هشام بن عروة عن أبيه عروة عن
عائشة أم المؤمنين رضي الله عنها ، أن الحارث بن
هشام رضي الله عنه ، سأل رسول الله ﷺ ، ف قال ، يا
رسول الله ! كيف يأتيك ال وحي ؟ ،
،، ف قال رسول الله ﷺ ، أحيانا يأتيني مثل
صلصلة ال جرس ، و هو أشده علي ، ف يفصم عني ، و
قد وعيت عنه ما قال . و أحيانا يتمثل لي ال ملك
رجلا ، ف يكلمني ف أعي ما يقول . ع
،، (-) قالت عائشة رضي الله عنها ، و لقد رأيته
ينزل عليه ال وحي في ال يوم ال شديد ال برد ، ف

Figure 4
Arabic window in Bayan.

Figure 4 shows an Arabic window which displays Arabic text information. Note that the text has displayed the proper "tashkeel" for every word of text. The figure shows the main options allowed by the system. Currently the Text_unit option is activated. The Text_unit object called "Bukh7.new" is displayed. The other options are document, query, learn and exit options. The document option, for example, allows the user to manipulate and create documents.

6. The Arabic word manager

This manager is responsible for deriving words from their roots and reporting the resulting word to the requesting procedure. This manager must know the rules for word derivations in Arabic as well as have the ability to learn new rules. This feature is built into the word manager because all rules of derivation that can apply to a word may not be known to the user when a word is added to the system. This feature will allow the user to add additional rules later if needed. Figure 5 shows the major components of the word manager.

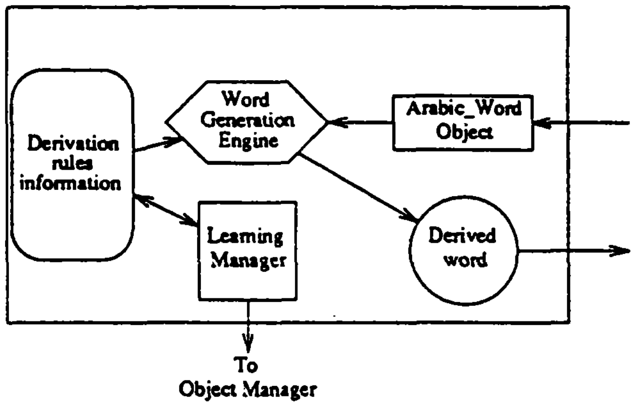


Figure 5
Word manager components

This manager allows a database administrator (DBA) to enter what is known as the teaching mode. In the teaching mode, the learning manager is called and then the DBA can add new words to the vocabulary of the system, add new derivation rules and specify which words it applies to. In the teaching mode, the DBA can modify existing derivation rules for the roots of an Arabic word.

7. The learning manager

This part of the word manager can learn new words. It takes information about a word and adds it to the object manager. It is important that this manager only be used by someone proficient in Arabic as incorrect information can be harmful to the database since Bayan stores words as objects rather than actual characters of a word.

Derivation rules are defined for each word by use of the learning manager. This is the only way to add a word to the vocabulary of the system. Arabic words are added to the database objects through this option after obtaining the required information to form a complete word object and to check if the word already exists.

The learning manager uses an interactive window-based interface that prompts the user to supply the required information. This process guides the DBA by requesting the needed information before accepting the new word. The DBA is responsible for the correctness of the information in the system.

The learning manager can also accept new derivation rules. Rules can be specified by the DBA using the proper format. Rules can be added to the knowledgebase of the learning manager and are used by the word generation engine.

8. The word generation engine

The word generation engine interprets derivation rules and executes commands encoded in the derivation rules. It is like a small computer executing a limited instruction set. It will take a root as an input and produce the newly generated word (if any) as output. If there is an error, the same root will be returned with a flag indicating the presence of an error. Errors can range from a null root to the wrong number of characters in the root and so on.

8.1. Derivation rules

All the rules are the same for each root class of words. Therefore, they need to be stored only once; afterwards, the word generation manager can generate the desired word from its root. It should be noted that rules are completely different for three character roots, four character roots and five character roots.

In general, a derivation is performed by adding characters and/or ehanging the "tashkeel" of the word. The encoding of the derivation is as follows:

<digit>
concatenate character number <digit> of the root adding the following "tashkeel". Where each <digit> will be followed by the proper "tashkeel".

<character>
concatenate <character> where character represents an Arabic character with its "tashkeel".

An example of what the derivation rules look like is the following:

مَ ١ْ ٢ُ و ٣َ ا نِ

مَ ١ْ ٢ُ و ٣َ ةٌ

مَ ١ْ ٢ُ و ٣َ تَا ن

Arabic derivation rules, however, do not only involve the concatenation of characters at the end of a word. Characters may also be added to the middle, beginning or end of a word to derive a new word.

8.2. An example:

Given the following:

Root = نَصَرَ

Rule = مَ ١ْ ٢ُ و ٣ُ و نَ

Result = مَنْصُورُونَ

The steps for this result is shown in the following table:

Step number	Resulting word so far	Comments
1	مَ	the new word starts with the arabic character 'Meem'
2	مَنْ	letter number (1) of the root word to be added.
3	مَنْصُ	letter number (2) of the root word to be added.
4	مَنْصُو	letter (waow) to be added.
5	مَنْصُورُ	letter number (3) of the root word to be added.
6	مَنْصُورُو	letter (waow) to be added.
7	مَنْصُورُونَ	letter (noon) to be added.

9. Morphological analysis of Arabic

A morphological analyzer for Arabic words is in the process of being built. It will be able to provide the following information concerning Arabic words: recognition of different parts of a word, the root of a word, and affixes attached to words.

The following information is required for the morphological analysis of Arabic words:

(1) The roots of words along with all the derivation rules that can be applied to them;

(2) The derived word and root links (which can be performed by a lookup table or by a smart algorithm) which takes a word and returns its root;

(3) A list of all possible affixes that can be attached to a word and that are not covered by the derivation rules; and

(4) A list of words that are not derived from roots, such as tools (e.g., "from", "to", "on" and so on).

10. Scenario

A user will create a text_unit object using bayan's editor. Figure 4 shows an editing session of a text_unit object called 'bukh7.new'. External files can also be imported to Bayan's editor using the "Load" option. An option can be activated by clicking on the button which represents the command. The updated Text_unit can then be stored in the database using the "Store" option. The user, when done, can ask Bayan to incorporate the new or modified Text_unit object into its final database. Bayan will check the Text_unit object for new words, index the words, or it can activate the learning manager if a word is not known to Bayan.

The user can specify the information which Bayan's learning manager prompts him/her for. This will include the specification of which class of Arabic words this word belongs to. The user will also verfiy the derivation method used to derive this word from its root. Additional information may be requested to determine affixes added to the word. If the user made a mistake in the word, he can go back and modify the Text_unit object using the editor, then ask Bayan to continue its operation. Bayan will add this information to its database. The user can query that Text_unit object by words that appear in it.

Documents can be built using the "Document option" which can be selected from the main menu. The user will be required to enter the elements which make up documents. These elements may include Text_unit objects and/or other documents.

The user can make queries about words within document objects. The result of the query will be a list of the document objects in which the word appears. The user can then specify that he wants to see the Text_unit object level and the Text_unit objects will be displayed. At the Text_unit object level the user can browse or modify these objects. If he modifies a Text_unit object he would need to ask Bayan to update its database as explained above.

Queries can also be made about Arabic words, their roots, or their derivation methods. Queries can ask Bayan to show examples of words derived from the same root, or even words of similar meanings.

11. Conclusion

Bayan is a starting step for more advanced research regarding Arabic. It points to the possibility of more advanced studies about the language, such as natural language processing based on Bayan's environment. For now, though, many persistent problems have been addressed, such as the design of an Arabic character set including the "tashkeel", an Arabic keyboard, word-generation algorithms and Arabic word generation methods from the roots of words. At the present time, an Arabic text data-base, which utilizes the representation and algorithms designed in Bayan, is being built. In addition, a built-in morphological word analyzer is also under construction.

Refrences:

[AHME86] Ahmed S. Ahmed, *'About retrieving Quran in Othmani script using computers'*, 9th national computer conference in Saudi Arabia 1986.

[ALSA86] Al-Saleh Mohammed, *'Standards for Arabic character in information'*, 9th national conference in Saudi Arabia, 1986.

[BECK87] Joseph D. Becker, *'Arabic word processing'*, Communication of the ACM, July 1987.

[BOOT86] L. Booth, et al, *'Arabization of an automated library system'*, 9th computer conference in Saudi Arabia, 1986.

[HASS87] Mohamed Hassoun, *'Conception et relisation d'une base de donnees pour les traitements automatique de la langue arabe'*, 10th computer conference in Saudi Arabic, 1987

[TAYL86] Murat Tayli, et al , *'Intelligent Arabic workstation'*, 9th computer conference in Saudi Arabia, 1986.

Responsa: A Full-text retrieval system with linguistic processing for a 65-million word corpus of Jewish heritage in Hebrew

Yaacov Choueka
Bar-Ilan University, Ramat-Gan, Israel

Combining sophisticated elements of morphological processing, pattern matching, "local feedback" and "short-context files" with a rich repertoire of tools for query formulation, browsing and display, coupled with special data structures and algorithms for a fast response time, the Responsa Full-Text software, operational now with a corpus of 65 million words of Rabbinical Hebrew, is specially well-adapted for processing complex queries in large textual databases in general, and in Hebrew (and Semitic languages) in particular.

1. Background and highlights

The Responsa Project (RP) -- recently incorporated in the more comprehensive and ambitious Global Jewish Database (GJD) -- is a complex network of research, development, applications, service and dissemination activities in the fields of information retrieval, computational linguistics, text processing and Jewish studies. In its context was built one of the very first full-text retrieval systems, and one of the largest corpora in the humanities (certainly the largest in Hebrew).

Conceived by A. Fraenkel in 1965-1966, who also directed it till 1975, the Responsa Project was launched in an operational batch mode [4] in 1967, as a joint endeavor of the Weizmann Institute of Science (Rehovoth) and Bar-Ilan University (Ramat-Gan), in Israel. In 1974/75 the project was completely relocated at Bar-Ilan to become formally part of the Institute for Information Retrieval and Computational Linguistics (IRCOL), which was created at that time. The author, who was part of the RP research team since its inception, has acted as Head of IRCOL and Principal Investigator of RP since 1975.

In 1980 the on-line version of the software was launched, and the highly positive impact of the concept convinced the developers to enlarge the scope of the Responsa Project -- originally limited to the Responsa Rabbinical literature -- to encompass the entire spectrum of the Jewish Heritage in an ambitious Global Jewish Database that would computerize -- so to speak -- the entire culture of the Jewish people. The idea crystallized in 1983, with the establishment of a three-year endowment fund for this purpose by a group headed by T. Klutznick in Chicago, and since then, classical writings in Hebrew are being constantly added to the database, subject of course to available funding.

Besides the basic components of a large database of Hebrew texts and of a sophisticated full- text on-line software (with unique linguistic capabilities) for the processing, retrieval, display and printing of documents and text-passages from the database, the project was also involved in research activities in the relevant fields of information retrieval and computational linguistics, and in the development of the infrastructure needed for supporting a large network of workstations connected to the system in Israel and abroad, and in servicing users from academic, legal, and rabbinic circles in processing queries in batch and on-line mode.

The special nature of the language of the texts (Hebrew, with its complex and rich morphology) and the special nature of the database (Rabbinical writings spanning one thousand years) forced the developers to face and solve problems that usually are not present in similar systems with English or with modern texts. In this short report, however, we restrict ourselves to describe in some detail only two of the above-mentioned components, namely, the database and the software. The latter will be described mainly from a user's point-of-view; thus, no algorithms, files or data-structures will be detailed. For these the reader is referred to the technical literature given in the references and to a more comprehensive "List of Publications" available from IRCOL.

Some of the software distinct features and highlights are: a rich and varied set of tools -- including proximity and boolean operators -- for query formulations; a flexible and dynamic display and browsing component for scanning large sets of documents in a short time; a complete morphological component for the automatic generation of all linguistically valid morphological variants of any term in the query; a rich pattern-matching module; tools for locally and dynamically creating, updating, editing and deleting "personal thesauri" and embedding them in the query; a "short-context" tool to help disambiguate highly frequent and ambiguous words that are part of the query; and an on-line "local feedback" tool that suggests to the user more synonyms -- or rather "searchonyms" -- to the terms appearing in the query. All of this in the context of advanced algorithms that assure very fast response time even for complex queries and in long-distance communications environments.

2. The database

The database of the GJD project consists today of some 65 million words of running text in Hebrew, organized in several distinct corpora. The largest of these is still the Responsa corpus which account for about 75% of the size of the GJD database; this corpus will therefore be described in some detail. A comprehensive and detailed list of all works that are parts of the database, including bibliographical details (author, edition, etc.) is available from IRCOL.

The Responsa literature, written mainly in Hebrew and Aramaic but occasionally containing expressions or even whole passages in vernaculars like Arabic, Ladino, German, Yiddish, English, etc., is an impressive collection of rabbinic "queries and responses" spanning many centuries, and originating from a large number of countries. The queries were posed by individuals and communities from all over the world to the outstanding Jewish Rabbis and authorities in each generation, and the decisions rendered became valuable precedents for Jewish Law, many of them being subsequently incorporated into the Codes. The Responsa literature is recognized as an invaluable storehouse of information of interest to scholars in many areas, such as law, history, economics, philosophy, religion, sociology, linguistics, folklore, etc. Personages, realia, geographic sites, scholars, wars and kings, together with those minutiae which bring the life of a community into sharp focus -- birth, marriage and death, recipes, taxes, medical practices -- all these and more provide the scholar with a unique opportunity to recapture Jewish life (and life in general) with an accuracy often unobtainable from other sources.

Interestingly enough, many topics that seem to be related to modern sociological and economic conditions such as ecology, devaluation of currency, minority rights, the right to strike, asylum claims for criminals and suspects, autopsy and euthanasia, were clearly described, debated and decided in this literature hundreds of years ago. Also, many of the Responsa recently formulated deal with current problems of genetic engineering, robotics, space astronautics, computerized devices, and the like. Finally, the legal material contained in the Responsa literature -- which is from this point of view nothing but a case-law compendium of precedents -- is of special interest, since it reflects in revealing details the operation of one of the oldest applied legal traditions in the Western World -- a system and philosophy of law that have been applied in diverse countries and under the most varied conditions ranging from predominantly agricultural and rural societies to urban, commercial and industrial states.

This remarkably rich legal and historical material has still to be systematically cataloged. It is estimated that about 1000-1800 authors have contributed to it, producing some 3500 printed volumes (collections) of Responsa totaling more than half-a-million (probably closer to three-quarters of a million) documents, originating from a large number of countries, mainly from Europe (especially Eastern Europe), North Africa, the Middle East and some parts of Asia. While the earliest responsa can be traced back to the Gaonic period in Babylonia (sixth to tenth centuries), it should be noted that for the last thousand years, responsa books have been continuously written and published, and this activity will probably accompany the Jewish people in the foreseeable future.

As of today, the computerized responsa corpus contains the full-text of 252 volumes of the most important Responsa collections, totaling about 49,000 documents with 50 million words of Hebrew. About thirty countries of origin are represented in this database, among them: Morocco, Egypt, Algeria, Turkey, Iraq, Syria, Greece, Yemen, Palestine, Tunisia, Lybia, Poland, Hungary, Russia, Galicia, France, Germany, Austria, Spain, Provence, Italy, and the United States. Also, every century as well as every school of thought and influential centers and authors from the tenth century downwards are represented in the database. Besides the Responsa corpus, the database contains, as the nucleus of the envisaged GJD, about 14 million words of many classical works of Jewish heritage, of which we mention: the Bible; the 36 Tractates of the Babylonian Talmud (5th century), including the famous Rashi commentary (11th century, France), in full; the Haggadic Midrashim, a collection of homiletical commentaries and expositions about the underlying significance of Biblical texts; a few Medieval Biblical commentaries; and Maimonides' Code (11th century). The complete works (20 volumes) of the modern Israeli writer and Nobel Prize winner S. Agnon are also included, as the nucleus of a modern Hebrew literature corpus.

This complex database is highly structured in a hierarchy of levels, as per the following scheme. The GJD is divided into several *Areas* (Jewish studies, Modern Hebrew literature...), each area containing different *Corpora* (Responsa, Talmud, Bible...), each corpus composed of different *Authors*, who may have several *Volumes*, composed of *Documents*, which are divided into *Paragraphs*, *Sentences*, *Words*, and *Characters*. Each one of these levels is separately indexed and can be directly accessed by the display and searching modules.

3. General Setting

The system (software and database) is installed on Bar-Ilan University's IBM mainframe, and is running under the MVS/TSO Operating System; it can be accessed locally through any standard IBM 3270-type terminal with Hebrew options. Outside Bar-Ilan University, it can be accessed using an IBM-compatible microcomputer, a modem, a telephone line and a specially developed proprietary communications diskette. The diskette simulates the IBM 3270 terminal on the PC, makes all the necessary protocol conversions between the asynchronous telephone communications and the synchronous ones required by the IBM communications software, while highly compressing the data sent and received. After an initial one-time setting, just inserting the diskette will automatically make all the necessary dial-ups and give all the required different passwords and account numbers, leading the user directly to the heart of the Responsa system. No black boxes, protocol converters or cards of any sort are needed.

The various functions of the systems are activated by touching the appropriate Program Function (PF) keys. Besides the standard PFs for *Help, Previous Page, Next Page, Print* and *Exit,* the following PFs are available:

Database Selection: For selecting the DB to which all subsequent search, display and browse functions will refer (until another DB is later selected); *Bibliographies:* Displays upon request biographical/bibliographical details on the works/authors included in the DB under investigation; *Text display:* For displaying any document (or part of a document) of which the exact references are known. Since different works usually have different internal referencing systems (some are subdivided into parts, chapters, and paragraphs; other are referred to by pages and sections; references are made sometimes by letters, other times by numbers, still others by Roman numerals), etc., and since our approach is to let the user feel comfortable with the system and use the same type of references with which he and his colleagues are familiar, we had in fact to devise a specific appropriate menu for each and everyone of the hundreds of works included in the GJD.

Note that once a portion of text from the DB is displayed, no matter why and by which function, the user can freely navigate in the DB, asking for any other section or paragraph of the displayed document (including *first/last/previous/next page/paragraph*), any other document of the same author/volume, etc. A title line at the top of the screen keeps the user informed about his exact location in the DB at every moment.

The following important PFs will be discussed in more detail in the next sections: *Search,* for formulating a query and displaying its results; *Thesaurus,* for the dynamic creation and editing of local thesauri to be included in the search; *Short-context,* for helping the user resolve ambiguous terms in the query; *Feedback,* to suggest to the user additional *searchonyms* (synonyms, related terms) to his query's terms; and *Morphological module,* that gives an on-line accurate and complete morphological analysis of any word (form) in Hebrew. Apparently, the Responsa System is unique (among other full-text systems in any language) in having the last three functions available in an operational retrieval system with a large corpus.

4. Query formulation: methodological considerations

As is well-known, the basic assumption underlying a full-text retrieval system is that the original text of the document is fully available on the computer and searchable by the software; the user frames his query, in the most basic form, by giving any word or phrase (= sequence of words) which he thinks would adequately represent his topic, in the sense that every document that contains this phrase -- and only such a document -- is relevant to the query and should be retrieved by the system. The specific language and corpus with which we are dealing here, however, and our personal view of the"philosophy" of full-text systems dictate some methodological considerations in which the Responsa system might differ from some of the other full-text ones:

a) In many of the systems operational in English, French and other European languages, *common* words (i.e., function or grammatical words) such as *the, with, and, from,...* are not searchable on the ground that they are *devoid of any information content..* In the Responsa System, however, each and every word (in fact, letter) of the text is searchable: first, because we believe that even common words can be of crucial importance when combined with other terms into meaningful phrases; and second, because most of these words (in Hebrew) are ambiguous and homographic with other *content* words (*if/mother; with/nation; but/mourning, etc.*) and thus cannot be omitted from the searchable text. The storage and processing-time problems are algorithmic in nature and should be handled adequately by specially-designed components of the system.

b) Usually a term in a query is a string of characters that has to be matched exactly by its counterparts in the text. In our case, however, a term would, more often than not, play the role of a representative of a whole family of "variants", containing tens, hundreds, thousands, or even sometimes tens of thousands of elements (= strings of characters). This may happen because of several reasons of which we mention here four (cf. the appropriate sections below for more details): *morphological processing,* where a lemma, say, has to be replaced by the family of its morphologically related variants; *pattern matching,* where the string -- with special "operators" embedded in it -- stand for a pattern of characters that has to be checked against every word in the text for a possible match; *thesaurus processing,* where the term is in fact the name of a whole family of related terms that have to be substituted for it; and *feedback process,* where a set of "searchonyms" is suggested by the system as a substitute for a given term of the query. This way a simple query of the form "look for the occurrence of *nuclear, explosion, dessert, kill* in the same sentence", becomes a complex combinatorial problem in which four very large sets have to be matched and processed at the same time. Sophisticated algorithms had therefore to be developed to handle such situations [7], while keeping the requirements on internal storage and processing time within reasonable bounds.

c) Notwithstanding the remarks made above, it is important to note that in the general philosophy of our approach, the system does not force upon the user any pre-defined concepts or procedures (such as pre-established *suffix omission* or *term expansion by thesaurus* components), but gives him always freedom of choice, besides establishing of course *default options* that are activated whenever he chooses not to interfere with the system's standard course of action. When large families of variants are generated from some of the terms in the queries, because of the reasons just mentioned in b), the user can either silently agree with the automatic inclusion of all the generated terms in the search process without even being aware of these generated forms or their numbers (and this is indeed the default option), or he can ask for the generated families to be displayed for his inspection before the search is done, in which case he can select from the displayed forms those he wishes to include in the search (or, if it is easier, to delete those that he wishes to omit). Thus a novice or casual user can conveniently rely on the default options, while a more sophisticated user can have control, if he wishes so, over each and every phase of the search process.

5. Query formulation: basic forms

In its simplest form, a query is a sequence of consecutive words that have to be looked for in the text, such as: **nuclear energy, court of appeal, testimony under oath.**

Note that common *(grammatical)* words can also be part of the query, and that sentence boundaries in the text are considered as phrase delineators. More powerful tools are available however to the user to formulate his query: proximity operators, direction specification, negated distances and alternative terms. Instead of describing, however, the exact syntax required, we just give a (somewhat artifical) example that clarifies the multiple tools available. Asking for

nuclear_atomic bomb_weapon (-3,5) desert (-10,10) -Nevada

means: any occurrences of **nuclear** or **atomic** that are immediately followed by **bomb** or **weapon,** in the proximity of which (from 3 words to the left up to 5 words to the right) **desert** occurs, which is not followed or preceded (in up to 10 words on each side) by **Nevada**. (Note that Nevada can occur somewhere else in the document or even in the same sentence.)

Qualifying a query with the operator **sentence** in its front, will cause the distances to be measured in units of **sentences** in the same paragraph rather than in units of **words** within the same sentence. Similarly for the operator **paragraph**. Thus

sentence: A (-1,1) B (0,0) C

is satisfied exactly when **A** and **B** are in the same or in adjacent sentences and **B** and **C** are in the same sentence.

Note that all of the tools mentioned above -- positive and negative distances, local disjunction, negated distances, and **sentence** or **paragraph** operators -- can be combined in one query, and will be interpreted accordingly. For example,

paragraph: terrorists (0,0) bombs (-1,1) -Lebanon

will be satisfied by any paragraph mentioning **terrorists** or **bombs** and such that **Lebanon** is *not* mentioned in that paragraph or in an adjacent one.

We finally note that any word (rather, any string of characters) that occur in a query as described in this section, can be made, if needed, to represent a whole family of hundreds or more related terms, that are automatically associated with it by algorithms for pattern matching, morphological processing or thesaurus expansion (as instructed by the user). Thus, families of words can be manipulated with positive and negative distance operators, negated distances, local disjunctions, and **sentence** or **paragraph** operators, exactly as is done with words. This is detailed in the following sections.

6. Generating families of terms: pattern matching

When required, a form in a query can represent, instead of a formal string to be exactly matched with its occurrences in the text, a *pattern* to be looked for in the text, where the pattern can contain mixed sequences of characters and special symbols, to be described presently. This can be very useful when looking for foreign names of persons or places, for technical terms, for spelling variants and the like (especially in a corpus such as the Responsa which spans long periods of time and contains different layers of the language), in international media publications that cover events in different countries and languages, in technical literature (chemistry, biology, pharmaceutics), etc.

Some available symbols are: the *don't-care symbol* ! which will match any single character; the *alternative string of characters* symbol / which represents an alternative between different possible strings in the pattern; the *variable-length don't-care* symbol * which will match any sequence of characters of arbitrary length (including the empty sequence of length zero), and which can be embedded at the beginning of the pattern *(left-hand truncation)*, at its end *(right-hand truncation)*, or anywhere in the pattern *(middle truncation).*

Of special interest is the *grammatical affix* symbol #, which when occurring at the beginning/end of a pattern, stands for a predefined list of grammatical prefixes/suffixes respectively. For English, such a prefix list would contain, for example, *un, re, dis, inter, ...* and a suffix one would contain *ed, ing, ment, tion, able, ...* . Thus **car#** would retrieve *cars* but not *careful,* and **#order** would retrieve *reorder* and *disorder* but not *border* . Such lists are specially helpful for Hebrew and similar Semitic languages where prefixes and suffixes are very common. (The two lists for Hebrew contain a few tens of affixes each.) This operator reflects a trace of linguistic processing, but it is still, really, a formal pattern-matching one; indeed **car#** would still retrieve *caring*, even though *caring* is related to *care* and not to *car.* Tools for true linguistic processing will be described in the next section.

Finally, boolean operators (**and, or, but not**) can be imposed on substrings in the pattern. Again, we content ourselves with one actual example. Looking for variant spellings in Hebrew for Venice, we first asked for *V*NZI*, getting 156(!) different forms, most of them being indeed variants of *Venice,* but many others being irrelevant, containing P, K or G at the beginning or end of the word. Asking then for *V(~P,K,G)*NZI(~P,K,G)*, a set of 85 variants was retrieved, all of them indeed pertinent to the search. Thus, a couple of minutes of intelligent dynamic interaction with the system dramatically enhanced its performance for this specific query.

Inclusion of a pattern-matching string as one of the terms in the query will cause the system to search for all forms found in the text that satisfy the pattern, and to substitute them automatically for that pattern in the query. This is not an easy task, specially for a corpus such as ours that contains more than half-a-million different forms. Insisting as we do on an on-line response precludes the possibility of a simple linear search on this list of different words of the corpus to find the forms that indeed satisfy the pattern, and advanced algorithms had to be developed to perform the task [3].

7. Generating families of terms: morphological processing

When a user is asking for all text passages mentioning *spy* and *expulsion* in the same sentence, he obviously does not have in mind these specific strings only, but any other morphological variant such as *spies, spying, expelled,* etc. as well. Yet, the possibility of sometimes expanding a term in a query into the corresponding family of its true morphological variants is usually missing from most (all?) retrieval systems with English texts. Two excuses are usually advanced to explain this omission. First, the number of such variants for a given noun or verb in English is small, three to five variants in general, and in any case less than ten for most words. Thus it is not unreasonable to ask the user to take care of these variants himself, adding them as alternative words in the query. Second, most such variants can be derived by adding a few well-known suffixes, so that the *don't- care character* symbol gives a simple solution to the problem, as in the case of **play*** which will expand into *plays, played, playing, player,* etc. The user can easily convince himself, however, that both these excuses are only partially true; and even if they are partially valid for English, they are certainly not so for French, for example (think of the verb *faire),* and they totally collapse for Hebrew, Arabic, and similar languages with a rich and complex morphology, where the number of variants can reach the hundreds and the thousands, and their spelling cannot be adequately developed by formal pattern- matching tools. In order to appreciate the complexity of the problem, and the need for an adequate solution, we now sketch a short review of the pertinent characteristics of Hebrew morphology.

The Hebrew alphabet contains 22 letters (in fact, consonants). Because of the lack of vowels, words are usually short (5 to 6 letters on the average), and it is quite rare to encounter words in a running text with a length of 10 letters or more; no word-agglutination (as in German) is possible.

Hebrew is a highly inflected language with a rich derivational pattern, and this can be best demonstrated through three figures. The total number of entries in a modern and comprehensive Hebrew dictionary does not exceed 35,000 entries (including some 3,500 international loan-words such as *bank, symphony*) derivable from about 4,000 *roots* of three to four letters. On the other hand, the total number of different Hebrew *words* (here defined as strings of characters which are meaningful and lexically correct units of Hebrew) is estimated to be about *fifty to one hundred million* . Thus, there is a gap of several orders of magnitude between the number of dictionary entries (= lemmas) and the number of words that are generated from them by inflection and derivation. (The corresponding numbers for English are estimated to be in the order of 35,000 "roots", 150,000 dictionary entries and one million forms).

Briefly, a noun or an adjective can appear in up to six or seven different variants (singular/plural/dual, masculine/feminine, and *construct* state); ten possessive pronouns *(mine, yours,...)* can be suffixed to each noun, and different (about thirty) combinations of prepositions and articles (*the, in, from,* etc.) can be prefixed to it. For a verb, the pattern is more complex. It can be conjugated in up to seven modes, four tenses and twelve persons; accusative pronouns (*me, you, him,...*) can be suffixed to transitive verbs, and prepositions can be prefixed to it (as with nouns). The number of variants for a nominal form is usually in the hundreds and can even reach one thousand, and for a verbal form can reach a few thousands. It should be emphasized that during the derivation process outlined above the original dictionary-entry form may undergo quite a radical metamorphosis. The Hebrew single word representing the phrase *and since I saw him* (10 letters) contains only two letters in common with the infinitive *to see* ; the word in Hebrew for *computer* has 1024 variants with twelve different leading letters, and many more thousands have to be added if we want to account also for the verb *to compute* .

Given the complexity of the derivational patterns described above, it is clear that the morphological component of retrieval systems for Hebrew texts cannot be neglected. Indeed, in a few experiments conducted on our database, in which some queries were processed both with and without the morphological component, it was found that the queries in which no linguistic expansion was performed, only 2%-10% of the relevant material were retrieved. On the other hand, no ordinary user will have the necessary linguistic knowledge and methodical reasoning, not to mention the time, energy and patience, needed to manually include all the different possible variants of the required word in the query, and the tools of the pattern-matching component are certainly not adequate for this purpose.

It was therefore necessary to include in the system an automatic counterpart of Hebrew morphology, and such a module, adequate for on-line processing, was indeed developed in the mid-seventies and incorporated in the Responsa system around 1980. The tools created can perform a full and correct grammatical analysis of any given Hebrew word, as well as find all linguistically valid variants of any given term in a given corpus. Basically the tools consist of a specially codified dictionary of Hebrew together with all the valid rules of conjugation and derivation, and exceptions to the rules. The interested reader can find more details in [1]; here we only describe the tools from a user point-of-view.

To every linguistically valid form we assign a *dictionary-entry* lemma, which is usually the singular masculine variant for a nominal form and the singular masculine third person with past tense for a verbal form (this form assumes the role of the infinitive in English), both stripped of any prepositional prefixes or pronomial suffixes. All lemmas with the same basic letters and the same semantic field are grouped together under one *root.* To give an example in English, the form *computed* and *computes* have the lemma *(to) compute,* and the lemma of *computers* is *computer* . On the other hand, *compute* and *computer* have the same root, which would also include the lemmas *computation, computerize,* etc.

Enclosing a term in a query in single apostrophes (**'compute'**) instructs the system that this is a *lemma* that should be replaced by all its variants occurring in the corpus. Enclosing it in double quotation marks (**"compute"**) indicates to the system that this is a *root* that should be replaced by all variants of all lemmas that can be derived from the given root. Finally, for a naive user who does not even know what is the lemma or the root of some term he is including in his query, enclosing the form in exclamation marks (**!compute!**) instructs the system to first analyze the given form and find its root(s), in order to replace it by all the derivative forms connected to this root. In English, for example, **'go'** would retrieve also *goes, going, gone, went,* while **"solve"** would retrieve, besides *solves, solving and solved,* also *solution,* etc. Finally **!thought!** would also retrieve *think* and *thinking* besides *thought*.

To sum up, when including a term in his query, the user has the option to consider it as a string of characters to be matched by an exact counterpart in the text, as a pattern to be replaced by all strings in the database that successfully match it, as a lemma to be replaced by all its derivatives and conjugated forms in the text, as a root to be first replaced by all corresponding lemmas and then by the variants of these lemmas, and finally as a word to be first analyzed and then replaced by the variants of its lemma(s).

Finally, it is not uncommon for a query to include four or more terms, each to be expanded into several hundred variants, with tens of thousands of occurrences corresponding to each term and its variants. Following (in Fig. 1) are actual examples of searches run on our corpus, detailing the number of variants for each term, and the total number of occurrences for these variants:

term	No. of different variants in the database	Total no. of occurrences of these variants in the database (rounded)
lighted	1	16
(to) 'light'	210	7400
candle	1	3200
'candle'	175	21300
and-he-testified	1	1
(to) 'testify'	336	53000
oath	1	4200
'oath'	261	59000

Fig. 1 Examples of morphological expansion

The computation resources needed to conclude the search in such cases can be enormous, and unless specially-tailored and quite efficient algorithms and data structures (such as described in [7]) are used, the search can be almost impossible to conclude on-line.

8. Generating families of terms: thesaurus expansion

The Responsa system does not include a global thesaurus that can automatically supply the user with synonyms or other terms semantically-related to the ones used in his query. For one, it is our -- as well as others -- experience that building such a thesaurus is a very complex, costly and time-consuming task that more often that not does not come to completion. We have also serious doubts about the efficiency and usefulness of such a *global* tool that is supposed to solve all problems for all users. In harmony with our general approach, we are more in favor of *local* thesauri, that can be dynamically created by real users for real needs, and then updated, edited, and tailored to specific areas. The system indeed supports this activity through the Thesaurus Program Function. Basically, the thesaurus contains "families", each family having a unique name and containing a set of related words. For example, a user searching frequently for dates may create a family called **month** and containing: *January, Jan., February, Feb.* etc. A user researching weapons may define a family **weapon** containing: *airplane, tank, rifle,* etc., while another one interested in flying objects may create a family **flying** containing: *airplane, bird, dove,* etc. Families can be very easily created, edited, deleted, or unified at will.

Since some of the members of a family can be names of other families, the user can construct hierarchies of related terms. Also, a word in a family can be a pattern or a linguistic lemma (or root), in which case, the word stands for the whole set of strings in the given corpus that satisfy the given criterion. Mentioning the name of a family in the query, instructs the system to expand this term into the entire family with the given name in the thesaurus.

As always, the user has full control on the generated sets of words, if he wishes so, examining them and deleting whatever seems to be irrelevant for the present query.

9. Storing and combining queries

All queries processed during an on-line session are given unique identifying numbers and are preserved for as long as the session is active, and can be stored in the user's workfile area for al long as he wishes. The user can combine stored queries by using the Boolean operators **and, or, but not,** forming thus a new query whose set of retrieved documents would be then constructed by the intersection, union or complementation, respectively, of the corresponding sets of the mentioned queries. Given for example the queries: 1. **testimony (-4,4) oath,** 2. **court of appeal,** 3. **paragraph: Missouri Court,** 4. **paragraph: Judge Williams**; a new query can be formed as in: 5. **#1 and (#3 or #4) but not #2.**

This approach of splitting a query into a few simple independent components that can be combined by Boolean operators into new, more complex ones, is in sharp contrast with the *hierarchical* one in which the only way to modify a query is by appending to it a new one that begins with one of the Boolean operators **and, or, but not,** thus restricting or expanding the set of documents retrieved by the initial query. In this way a query modification always refers to the last query constructed, and there is no way of retracing back false steps, of trying side issues, or of experimentally studying the effect of some modification without committing oneself definitively to that path. On the contrary, the adopted philosophy described here is well adapted to such modifications in search strategy, so typical of complex searches in document retrieval systems. Referring back to the example above, if query 5 proves unsatisfactory, maybe because component 3 is ill-defined, one can change query 3 to: 3. **sentence: Missouri Court,** and run query 5 again, or one can just ignore query 3 altogether and run: 6. **(#1 and #4) but not #2.** By using this approach the user is deliberately educated into avoiding long complex formulations that are difficult to understand and to modify, and, instead, to split his request into a few small "building blocks" that can be combined and recombined at will.

10. Presenting the results: the display and browsing component

One of the major lessons learned from our long experience with full-text document retrieval systems is that the user/system interaction -- in terms of query submission and results display -- is more often than not a multi-step, dynamic, conversational and exploring one, rather than a one-shot question - answering procedure. The browsing module in the Responsa system was designed to meet this situation. Limited space prevents us from elaborating on this aspect here; let us then just highlight some interesting points.

After the completion of the search, statistics on the number of retrieved documents and the number of hits will be displayed as a first item of information to the user, since we believe that the user strategy for scanning the results depend much on the number of retrieved items. Together with these statistics the system displays a set of *selection options* (by author, number of hits, etc.), which the user can trigger to reduce the number of documents to be displayed, if he judges it to be too large for his purpose. Once that selection is done, the starting point for browsing is a special KWIC (Key-Word-In-Context) of the occurrences of the hits in the retrieved documents. An example, adapted for English, for the query **testimony (-4,4) oath** is given in Fig. 2.

	Vol. IV 1983	
...The court should always consider a	testimony	under oath as a valid...
...specially that all these	testimonies	were given under oath...

...being under oath, he	testified	that the accused was...
...witnesses sometimes lie on oath, such	testimonies	should be taken...
	Vol. VI, 1985	
...I am a doctor, he	testified	and on oath to save life...
...no way to convince him;	testifying	under oath, he denied...
	Vol. VII, 1986	
...he refused to	testify	claiming that an oath...

Fig. 2 -- A KWIC sample

KWIC lines from different documents in the same volume are separated by a few dashes on the screen; this way a quick glance at the screen will be sufficient to assess what are the most promising documents. Exact references for each line of the KWIC can be asked for, and these will then be displayed at the beginning of the line (replacing there a few words of the KWIC).

Our experience with this type of output as a first display is excellent. One can scan large amounts (i.e. hundreds) of documents in a rather short time and get a good and reliable impression on the relevance/non-relevance of many of these documents. Several additional options are however available to the user in case he decides to examine the output in more depth. The user may wish, for example, to expand some of the displayed lines into larger chunks of texts, either because the given context is too small to be conclusive, or on the contrary because the item is clearly relevant and the user wants to see more of it. The user can then mark the lines he would like to expand and even specify how large a context he is asking for: document, paragraph, or just **n** sentences surrounding the examined line. Once the required context is displayed, the user can browse freely in the document asking for the next/previous last/first paragraph/page, and when finished, ask for the expansion of the next marked line, or redisplay the KWIC page in order to resume his research. Finally, the KWIC lines will normally be kept in a file for further processing or printing. The user can mark some of these lines for deletion -- if judged non-relevant -- and only the remaining lines will be kept or printed.

11. Short-context tools

This is one of the most useful tools available to the user to partially resolve the (morphological) ambiguity problem. When the user realizes that one of the terms in his query is highly ambiguous (such as *party* or *light* in English) and can generate a large number of irrelevant citations, he can first turn to the "short-context" component. Given an ambiguous term as a parameter, the type of short context required: *following* neighbor or *preceding* one, and the sort option required: alphabetically or by decreasing frequency, this PFtriggers the system to display the list of all *different* left or right neighbors (as requested) of the given term, together with the frequency of these two-word expressions in the corpus. Previous experiments on texts in French [6] have proved that in most cases a one-word context is sufficient to disambiguate an ambiguous word (at least for certain types of ambiguity), that the error rate in such a procedure is very small, and that indeed a relatively small number of frequent neighbors will account for a large proportion of the term's occurrences in the corpus. Although formal results are not yet available for Hebrew, our experience shows that the same characteristics are also true of Hebrew. Some experiments recently conducted by the author while visiting Bell Communications Research in New Jersey, U.S., on a New York Times corpus of ten million words gave similar results for English. Figure 3 shows, for example, a few lines from the lists of the right neighbors of *interest* and the left neighbors of *party* with their frequencies (as a two-word expression) in the corpus:

communist	party	489	interest rates	1593
opposition	party	41	interest payments	145
third	party	29	interest groups	35
birthday	party	28	interest income	16
dinner	party	24	interest charges	10
herut	party	10	interest costs	10

Fig. 3 -- Short-context samples

Scanning the list of contexts of the given ambiguous term, sorted by decreasing frequency, the user will usually be able in a couple of minutes to assess the appropriateness of many lines of contexts, and to mark for deletion those that are not relevant, probably eliminating in just one stroke thousands of non-relevant items. The search will now be restricted automatically to those documents that contain the given terms in the appropriate contexts only.

12. Feedback

When framing his query, the user may inadvertently omit certain synonyms or otherwise related terms from his query formulation. He might give, for example, *airplane* but forget *helicopter, jet, aircraft* or ask for *drugs, cocaine* but omit *opium, heroin,* or *LSD* . This problem can be solved partially using a good thesaurus. There are however two problems with this solution. First, comprehensive good thesauri are not so readily available or constructed; second, *synonymity* is a rather misleading term in this context. In information retrieval, one is not really looking for *synonyms,* i.e. words that are equivalent to his query's terms in a general linguistic sense, but rather for *searchonyms,* i.e. words that in the context of his specific search are indeed equivalent.

Most of the feedback methods developed in the last twenty years were based on the statistical analysis of terms distributions in specific documents and in the whole database, and used the so-called term-term and term-document matrices. Their results were rather mixed. New ideas were introduced in [2], where a concept of *local feedback* was defined, in which proximity constraints, local contexts and only relevant retrieved documents were taken into account, and positive results were reported there for that approach when tested in a batch environment on a small part of the Responsa database. This approach was lately incorporated in the on-line version, first on an experimental basis [8] and lately in an operational one. After running the search and returning a set of retrieved documents, the system receives as feedback from the user a list of those retrieved documents that are judged by him to be relevant. The system then processes this information and returns a list of up to twenty potential searchonyms for each term used in the query. Our experience with this component is very promising. The returned lists usually contains quite a few interesting searchonyms, sometimes in a totally unexpected and even startling way. In an actual example run on the system, asking for **smoking** in the proximity of **health**, the feedback process suggested the following searchonyms (among others): *cigarettes, cigarello, tobacco, inhaling, chronic, risk, bronchitis, coughing,...* . Moreover, the feedback component was used also successfully -- in a peculiar "negative feedback" fashion -- in order to enhance the *precision* of the system, i.e. to reduce the number of non-relevant items retrieved [8].

13. Conclusions

Full-text retrieval systems, once a mere curiosity, have now come of age, and can certainly be considered today a well-established methodology, with a large number of applications, specially in the domain of legal material, newspapers and magazines, scientific abstracts, etc.

It is quite surprising therefore to find out that most of the available and operational full-text systems of today are still lacking many of the features described above and developed essentially in the late seventies and early eighties. Automatic morphological analysis, complex pattern matching, proximity operators with *sentence, paragraph* and *negation* capabilities, dynamic thesaurus building, short-context browsing, local feedback and the like, should by now be standard components of any professional full-text system. Even when a full-text system -- like the Responsa one -- does include all of these features (and more) as standard capabilities, it is still to be looked upon as a *third-generation* system (the *first- generation* ones being the crude batch systems of 1966-1975 and the *second-generation* ones being the simple on-line systems of 1975-1982). Research and development efforts should be directed now to *fourth- generation* systems, in which artificial intelligence techniques would be used to dramatically enhance the performance of the retrieval environment while greatly simplifying the user-system interface, reducing it to an almost natural language dialogue. Time will show whether such expectations will be realized soon.

References

1. R. Attar, Y. Choueka, N. Dershowitz and A.S. Fraenkel, KEDMA-Linguistic tools in retrieval systems, Journal of ACM, 25 (1978), 52-66.
2. R. Attar and A.S. Fraenkel, Experiments in local metrical feedback in full-text retrieval systems, Inform. Process. & Management 17 (1981), 115-126.
3. P. Bratley and Y. Choueka, Processing truncated terms in document retrieval systems, Information Processing and Management 18 (1982), 257-266. See also Proc. of the 16th National Conference of IPA, Jerusalem, 1981.
4. Y. Choueka, Full-text systems and research in the humanities, Computers and the Humanities 14 (1980), 153-169.
5. Y. Choueka, M. Slae and A. Schreiber, The Responsa project: computerization of traditional Jewish law, in: Legal and Legislative Information Processing, (B. Erez, ed.), Greenwood Press, 1980, ch.18, 261-286.
6. Y. Choueka and S. Lusignan, Disambiguation by short contexts, Computers and the Humanities 19 (1985), 147-157.
7. Y. Choueka, A.S. Fraenkel, S.T. Klein and E. Segal, Improved techniques for processing queries in full-text systems, Proc. of the 10th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, New Orleans (1987), 306-315.
8. S. Hanani, On-line local metrical feedback components in full-text retrieval systems, M.Sc. thesis, Bar-Ilan Univ., Ramat-Gan, Israel, 1987 (Hebrew).

Iconic Communication: Image Realism and Meaning

Fang Sheng Liu, Ju Wei Tai

Institute of Automation, Academia Sinica
P. O. Box 2728, Beijing 100080, China

ABSTRACT: In iconic communication, an important problem is to visualize concepts such as data and operation with small images called icons.So the relation between the realism of an image and the meaning it represents is an important problem to solve. Motivated by Chinese character category theory, this paper discusses how to construct a visual iconic human interface with realistic icons. It gives a set of principles to make icons more realistic and accurate in representing meaning. Via our group of principles, an icon may form a situation together with some meaningful images. At the same time the group of principles introduces structures into an icon. The paper also discusses the interpretation and the evaluation of an icon. Finally, it gives an example of an iconic interface of a database system.

1. INTRODUCTION

The most often used meams by which people communicate is natural language. Natural languages are in some sense artificial laguages, made by people. They are natural because they are closer to our daily life.In fact, natural languages are different from each other. There are languages based on phonetic letters, like English, as well as languages based on ideographs, such as Chinese. Phonetic letters convey meaning by spelling words,while ideographic languages express meaning with pictographs or icons. So they have completely different origins. They are the same in the sense that they are both string languages. In an ideographic language, a character has no direct relation with its pronunciation, every character has only one syllable,but the character itself expresses a meaning.In many cases, not only in science and technology, but also in daily life, people find that the representation of meaning with graphics is more natural and understandable. Arnheim says that iconic representations are universal and easy to recognize[5].In human computer interaction it provides good human factors.

When we mention the representation of meaning with icon or iconic languages, we are in fact talking about two things, the image and its meaning, i.e. a dual representation of a generalized icon (meaning, image)[1]. So we are faced with at least three problems: how to construct the image part (image or picture for short in the rest of the paper) to express the meaning accurately; what is the relation between an image and its meaning; and the interpretation of a generalized icon (icon for short in the rest of the paper). Expressing meaning with icons is sometimes ambiguous [2]. Because an icon is a simplified picture, the simpler an image, the more ambiguous[3]. On the other hand, there is no commonly accepted universal set of icons [4]. This paper is focused on iconic communication with better images,that is to say,the creation of realistic icons and the relation between image realism and meaning.

The Chinese language is very successfull in expressing meaning with small pictures or images. Chinese characters are pictographs. They are the cream of the Chinese culture.Chinese character category theory called " 六书 " generates and interprets Chinese characters from small meaningful pictures and combines them to form new ones. " 六书 " says that there are six categories of Chinese characters.

Category 1:	(形象)	pictographic characters or pictographs;
Category 2:	(指事)	self-explanatory characters;
Category 3:	(会意)	associative compounds, which are formed by combining two or more elements, each with a meaning of its own, to create a new meaning;
Category 4:	(形声)	pictophonetic characters, with one element indicating meaning and the other sound;
Category 5:	(转注)	mutually explanatory or synonymous characters;
Category 6:	(假借)	phonetic loan characters, characters adopted to represent homophones.

This is the architecture at the level of Chinese character set. We will see the internal structure in a Chinese character below.

Chinese characters have been evolving from ancient forms that were small pictures into the modern forms, Chinese ideogram. The evolution consists of seven stages:inscriptions on bones or tortoise shells, inscriptions on ancient bronze objects, seal character (a style of Chinese calligraphy), official script in Han Dynasty (an ancient style of calligraphy), cursive hand (characters executed swiftly and with strokes flowing together), regular script, and running hand. For these and other reasons, there are often Chinese characters that have more than one writing form, i.e. ancient form, modern (simpified) forms,a variant form, and/or the original complex form. In the Chinese language, the original meaning of a Chinese charater means a concrete thing. But its extened meaning covers a larger area. In fact, human languages should denote concrete things first, and then abstract, even mysterious things second.

This paper got a lot of enlightenment from the Chinese character category theory.

2. SOME PRINCIPLES OF ICON CONSTRUCTION

In this section we discuss principles for making the image part more realistic and accurate in representing the meaning part of an icon. So, we focus on what appearance an icon should have to express a meaning, and the organization of icons.

2.1 Principle 1: Appearance Reference

No matter what concepts we are faced with, we have three classes of things: things that can be found in nature, artificial things made by man, and abstract things. To give a concept about a thing we can show its picture or image directly. This is virtually the most superficial and basic method. For a concrete concept,such as natural and artificial things,we may take their natural or external shape as the icon image. If a natural thing has no visible shape, we may take the graphic representation of its phenomena and/or behavior, etc as the image representing a meaning. Examples are shown in Fig.1. The image of an artificial thing may also take its external form.

In many cases, abstract concepts cannot be represented with an image as directly as above. The basic idea for describing abstract concepts is to make the abstract the concrete or to use the following principles, such as Principle 5 and 6. We will use the description of meaning by combination and/or similarity or analogy. In addition, we may use the artificial symbols that have been stan-

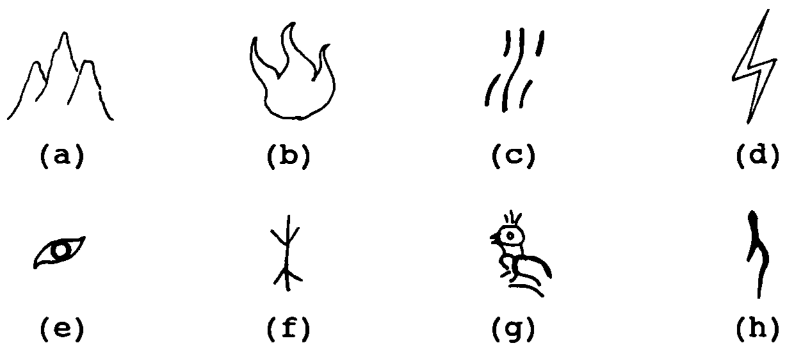


Fig.1 Visible Appearance as Image
(a) mountain (山); (b) fire (火); (c) steam, river, water (水); (d) electricity; (e) eye (目) ;(f) wood, tree (木); (g) bird (雀); (h) human being (人)

dardized or have become conventions.

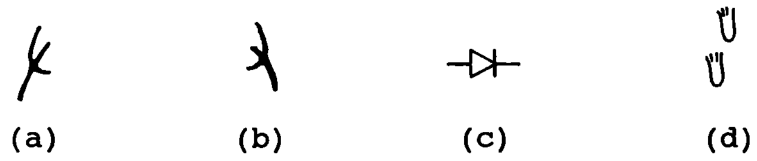


Fig.2 Representation of Abstract Concepts with Image
(a) left (opposite to right) (左); (b) right (opposite to left) (右); (c) diode; (d) walk (歩)

The image in Fig.2 (a) is a left hand drawn in a simplified style, so its meaning is "left", with the modern Chinese character in the right-most bracket of the legend.The image in Fig.2 (d) shows a pair of walking foots. Hence, the meaning of the image is "walk".

Further more, we may combine some small pictures with the image of existing icons made by Principle 1 or remove small pictures from them to create new icons. So these kinds of icons are a little bit different from those created directly from small pictures. Fig.3 gives two examples.

Fig.3 Concept Represented by Images
(a) fruit (果); (b) nest,bird's nest (巣)

In Fig.3 (a),the tree is full of fruits.This image can also be drawn as " ". So we get the meaning "fruit" from it. But we cannot get a definite meaning from the small picture " ". In (b), by combining " " with the icon for "tree", we get an image showing that birds are in the nest on a tree.So we have its meaning "nest,bird's nest".

2.2 Principle 2: Marking Emphasis

We can mark a special position or area of an image with marker to emphasize the "meaning",and to make clear its local semantics and its relation to other parts of the icon image. This is often done on an icon formed via Principle 1. Usually we use a point marker,arrow, circle, color, blinking icon, or a highlighted icon, etc to make the "meaning" evident to the user. There are some examples in Fig.4.

Fig.4 Making Meaning Evident
(a) root (本); (b) end,tip (末)

In Fig.4 (a), an emphasis marker is put at the root part of a tree. Hence, an icon expressing the meaning "root" is created. In (b), a point marker is put on the top part of a tree. Hence, an icon representing the meaning "end" is produced.

2.3 Principle 3: Reference by Similarity-Peculiarity

By similarity, we create half of an image of an icon,which shows the class to which the icon belongs or its relation that the icon has to a definite concept. The half may be pictographic. By peculiarity, we construct the other half of the image, which shows its peculiar or characteristic meaning or properties.

Fig.5 Meaning Represented by Concept Constraints
(a) female animal (牝); (b) male animal (牡)

In Fig.5 (a), the left half image is an ox head, meaning that the icon is used to describe animal-related concepts. The right half of the image is image for ewe, sow, mare, etc. Hence, the meaning is "female animal". In Fig.5 (b), the right half of the icon is a image for ram, boar, horse, etc. So the meaning is "male animal".

With this principle, more icons can be created to represent the meaning needed. In fact, this principle goes a step further than the above three. Many concrete things and abstract concepts are difficult to expresse with Principle 1 and other principles in the paper. For example, "fish" is a general concept. But we have thousands kinds of fish. If for every kind we create an icon, a large number of icons for "fish" will be produced. So we need a more efficient principle to solve the problem, i.e. Principle 3.With this principle,we only need to combine an icon that can show the features of a special kind of fish with an icon " " to produce an icon representing the concept for the special kind.

2.4 Principle 4: Borrowing from Other Representations

By borrowing,we express the images that are not easily expressed directly. The borrowed image may have a visible external shape and take the "meaning" as its property, usage, behavior, function, etc. Fig.6 gives two examples.

(a) (b)

Fig.6 Creating Icons by Borrowing
(a) direction control; (b) fragile

2.5 Principle 5: Extending the Meaning of Existing Icons

In some cases,we may extend the meaning of an existing icon to a larger but related area. This is of course a step further in iconic communication. But conventions may be needed. On the other hand, with this principle,we may make full use of the existing icons and avoid the unnecessary increase in the number of the existing icons.

For example, the meaning of the icon for "fruit" in 2.1 is extended to express the meaning of "result, consequence", while the icon for "limit" in 2.6 gets an extended meaning of "restrict".

2.6 Principle 6:Interpretation via Integration and Transformation

Combining two or more icons may form a new icon image. The resultant image is often a meaningful image that narrates a process, a situation, a result, etc. Like Chinese character generation, we have different subrules to guide the formation of a new icon image [4]. In Fig.7 are several examples. In Fig.7 (a),the left half is an

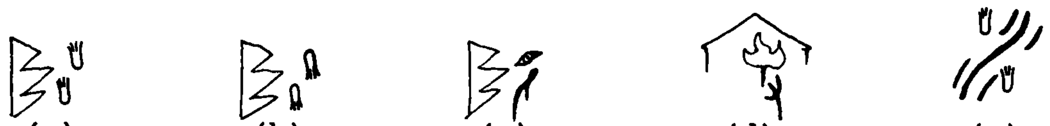

(a) (b) (c) (d) (e)

Fig.7 Meanings through Conceptual Interpretation
(a) ascend a height, go up (陟);(b) descend, fall, lower (降); (c) limit, bounds, limitation (限); (d) search (搜); (e) wade, ford (涉)

image for "mountain", the right half is an image for "walk"; therefore,the picture says that two feet are walking upwards along a mountain. That is to say, the meaning is "go up". In a similar way, the interpretation of the icon in Fig.7 (b) is "descend", and the meaning of the icon in Fig. (e) is "wade, ford". For the icon in

Fig.7 (c), we get the observation that a person " " turning his head is looking " " at the high moutain " ". Certainly his line of sight cannot go beyond the mountain.Hence we get the meaning "limitation". Fig.7 (d) gives an icon,the upper part of which is the simplified icon for "house",the middle part of which is fire,and the lower part of which is a hand. So the interpretation is searching with a torch in hand, that is, "search".

In Fig.8,we have a group of icons related to hand. In Fig.8(a), a hand in the upper part is picking fruit on a tree in the lower part. So the icon represents the meaning of "pick, pluck, gather".In Fig.8 (b), a hand in the right part is taking the ear in the left part. In ancient China, the icon had the meaning of "cut down (sb's) ear". Through evolution, it got the current meaning "take, get, fetch". Fig.8 (c) shows that a hand in the lower part is catching a bird in the upper part.So the icon has the meaning "capture, catch". The left part of Fig.8 (d) is a hand.The lower right part is mortar. The upper right part is a pestle. So the interpretation of the image

Fig.8 A Set of Icons with "hand" Icon
(a)pick, pluck, gather(采);(b)take, get, fetch(取);
(c) capture, catch (获); (d) insert, stick in (插)

in (d) is that a hand pokes into a mortar, that is to say, "insert, stick in".

2.7 Principle 7: Combination of Graphics with Text

The dual representation of a generalized icon can be extended to (meaning, graphic+text). "text" here refers to function words or key words. Sometimes the text part may be some symbols. In this way, the image part may be described vividly and more accurately. This principle may be nearer to Category 4 in some sense. For example,the meaning of image " "is "no parking", while the meaning of " ⚠ " is "warning".

2.8 Principle 8: Reference by Association and Comparison

For those icons that may have different meanings in different contexts, by attaching pictures with sharp contrast, we may make the desired meaning evident and obvious. The Fig.9 is an example.This is

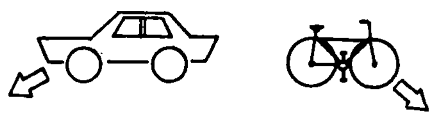


Fig.9 An Icon in Transportation Control:
car left ,bicycle right

an icon we often see on the street. It means different transport vehicles take different roads.

2.9 Principle 9: Reference by Amplification and/or Selection

Amplification means making the related part of an image evident and clear. Selection means selecting the image wanted from many images, this is in some sense consistent with Principle 2. We may, for instance, use a " ✓ " sign to select. The difference is that we have many images here.

This group of icon generation principles is not only efficient in creating primitive icons, but also efficient and effective in constructing composite icons. They can be used (in different combinations) to form different parts of an image in an icon. When the principles are applied to create an image, not only meaningful small pictures are put together, but also structures between them are built. A typical structure is an attributed tree[2][3]. Furthermore, the constructed image in real systems may be of higher realism, including external shape, greydegree, colors, and so on.

3. MEANING AND INTERPRETATION

The created icons for simpler and/or concrete concepts may have simpler images. Their interpreation is easy. But for more complex images, the interpretation is important. In this case, the composite images of a Chinese character,for example,may form a situation other than combination of irrelative small pictures representing meanings. In such a situation there are often subject, object and different adverbial modifiers. Also, a conceptual graph may be built to interpret the meaning.The interpretation is sometimes focused on the process in such a situation, sometimes on the results, sometimes on the extended meaning, and sometimes on other aspects. An example is shown in Fig.8, where "hand" is the subject of behavior.On the other hand, the interpretation has its own logic. The interpretation of icons for different fish is an example. In that case," 魚 " is used to represent the intension of the concept "fish". But by adding special features for a given kind of fish, we may make the new icon cover a narrower intension only focusing on the special kind.

We may make an observation that in many interpretations of created icons, some parts of an image may be changed to other small pictures without changing the meaning. For Chinese character generation only, the special small pictures in the image part are possibly used to represent a special meaning, since every Chinese character has its own original meaning and extended meanings.Also in old times many Chinese characters often had one meaning.After years' evolution only one of the characters is used and others are not used any longer. Generally speaking, we may also deal with such cases by applying the above principles to create icons in iconic communication.

The interpretation of icons in Chinese language requires background knowledge,such as knowledge about nature, society, and so on. As for Chinese characters, they were not created in one day. Some of them were created first,and then new icons for more complex concepts were obtained from images and existing icons. Some examples are illustrated in Fig.10. Fig.10 (a) shows an icon for "knife". Fig. 10

(b) is a picture in which a knife cuts a thing into pieces. So the meaning is "separate". Fig.10 (c) is the today's form of a Chinese character with the meaning of "break off with the fingers and thumb" . The left and right part of the character are two hands. The middle part is a character shown in Fig.10 (b). So the explanation is

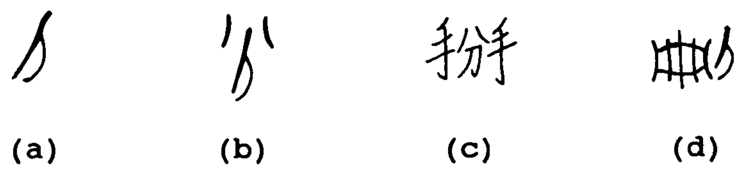


Fig.10 Example Icons for Complex Concepts
(a) knife, sword (刀); (b) divide, separate (分);
(c) break off with the fingers and thumb;(d) delete (删)

"separate with two hands", that is, the meaning mentioned above. In Fig.10 (d),the left part is an icon for "volume, book", since during ancient times a volume or a book made of bamboo slips was stringed together. The right part is the icon for "knife". Hence, the whole icon produces an interpretation that a knife deletes the unwanted part and holds the desired part. That is the meaning "delete".

4. AN EXAMPLE OF ICONIC INTERFACE IN A DATABASE SYSTEM

In computer systems, both data and operations can be represented with icons[2]. Suppose that we have a multimedia database system for private cars management,in which the owner's picture with other data such as his signature, and the picture of his car and other parameters are stored. In the database system, we have a data manipulation language based on < D, O >, where D is the set of data; O is the set of operations.

Note: O = { find next, insert, delete, update }.

Suppose that its visual iconic interface provides not only icons for data and operations but also iconic frames to combine iconized data and iconized operations to form new icons which are called iconic sentences. First, we iconize the system in Fig.11 to build an interactive human computer interface. The icon in (e) is a borrowed

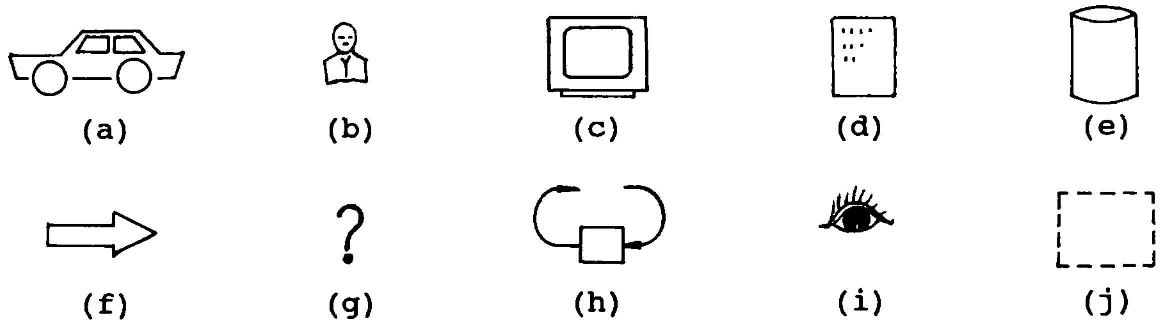


Fig.11 The Basic Icon Set for Example Database System
(a) Icon for cars and related data; (b) Icon for owner and related data; (c) Icon for screen; (d) Icon for "file"; (e) Icon for "database"; (f) Icon for "data movement"; (g) Icon for "condition"; (h) Icon for "loop"; (i) Icon for "find"; (j) Icon for "modularize"

denotation. The icon in (g) may be regarded as one by Principle 5. The other icons in Fig.11 are created by Principle 1. In the rest of the section,we discuss the construction of icons for data,relations, iconized operations or commands to communicate with computers, by the examples shown in Fig.12. Obviously, we can construct the icons in Fig.12 by applying the principles in section 2. In this iconic interface, there are two areas called working area and icon area. Icon and iconized objects such as iconized data are different. The latter is called an icon instance.Every time icon data is used (e.g. put into a "modularize" icon in an icon sentence, or assigned) or an icon command is executed in a working area, it becomes an icon instance. On the other hand, we can construct icons and put them in icon area to be used in the future.Data assigment can be implemented by form-filling."Modularize" icons make separate data or relations a whole unit for latter use (e.g. for data: "connect" operations,for relation: "and" and "or" operations). In an iconic sentence,the sub-icon in the innermost "modularize" icon is executed first, and then

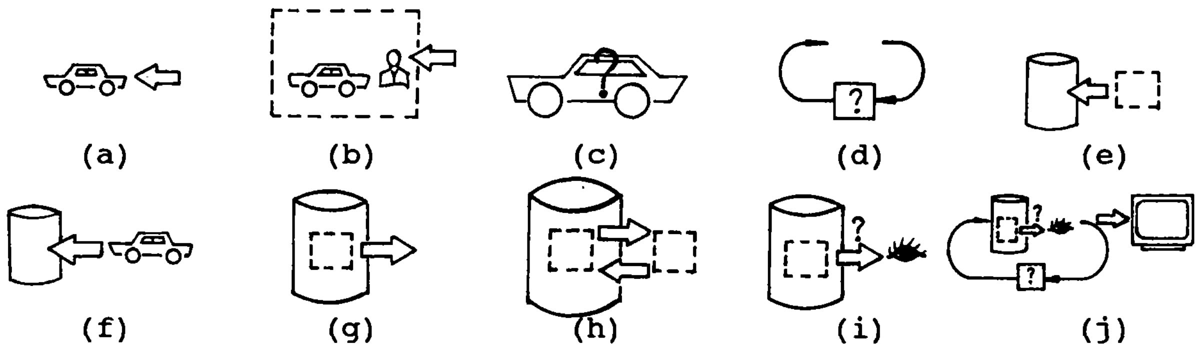


Fig.12 Iconic Data Manipulation Language
(a) data assignment to car; (b) data assignment to both owner and car; (c) condition about car; (d) loop with condition; (e) insert data in modular frame into database; (f) insert data about car into database; (g) delete data in modular frame from database; (h) update data in the modular in database icon with data outside; (i) find data according to the given condition;(j) find data according to the condition a definite number of times and then display the results on screen

the sub-icon in outer "modularize" icon is executed. The result of icon execution is data. In this way, new icons of data, relations and higher level operations can be constructed; therefore, an iconic data manipulation language is obtained.

5. CONCLUSIONS

There are many reasons for the ambiguity in iconic representations of meaning. One is the lack of conventions and standardization. Another one is that the meaning of iconic units is related to the situation and context in which the icon is interpreted. Still another is the lack of dynamic behavior in iconic representations.

The principles in this paper may be applied to primitive icons as well as composite icons.They can be used for both simple concepts and complex concepts, to represent concrete concepts and abstract concepts. They make the image part more realistic and closer to its meaning.

By studying the Chinese character category theory, we may better understand how people interpret icons and eventually develop a better psychology of visual iconic human computer interactions.

Chinese character category theory is closely related to Chinese culture, philosophy, history, geography, religions,and so on. So the principles can be misused. Although Chinese is based on ideographs, the Chinese language itself is still a string language. Our research goal is to develop 2-D (even 3-D) languages, since iconic language is universal, easy to understand, powerful in describing many 2-D problems and so on.It should be pointed out that iconic language and form language are often used together. Iconic communication and string language-based communication cannot replace each other.

The interpretation of an icon depends on the concepts as well as the conceptual structures that are represented in the icon. The evaluation of icon construction puts more emphases on the ease with which the icon is interpreted. The easier the interpretation is, the better the construction of an icon. If an icon is possibly interpreted ambiguously, the construction of an icon is a failure.

In the future, more work should be done on the interpretation of an icon, especially the conceptual interpretation based on the structures of meaning. This needs support from database systems and artificial intellegence, as well as 2-D computational liguistics.

REFERENCES

[1] S. K. Chang, Visual Languages: A Tutorial and Survey, IEEE Software Vol. 4, No. 1, January 1987

[2] F. S. Liu and J. W. Tai, Some Principles of Icon Construction, in Visual Database Systems , T. L. Kunii ed., Elsevier Science Publishing B.V., The Netherlands, 1989

[3] F. S. Liu, The Modelling and Analysis of Human Computer Interaction and Visual Representations, Ph.D. Dissertation, Institute of Automation, Academia Sinica, Beijing,China, August, 1989

[4] R. R. Korfhage and M. A. Korfhage, Criteria for Iconic Languages in Visual Lnaguages, eds. S. K. Chang et al., Plenum, 1986

[5] R. Arnheim, Visual Thinking, University of California Press,1969

[6] Xu Shen (author), Duan YuCai (explains), Exposition and Explanation of Chinese Characters, ShangHai Ancient Book Publishing House, 1988, 2nd Edition

[7] Lu SiMian, Four Kinds of Chinese Character Studies, ShangHai Educational Publishing House, June, 1985, 1st Edition

[8] S. L. Tanimoto,Visual Representation in the Game of Adumbration, Proc. of IEEE 1987 Workshop on Visual Languages, August, 1987, Linkoping, Sweden

[9] K. Furuya,S. Tayama,E. Kutsuwada and K. Matsumura, Approach to Standardize icons, Proc. of IEEE 1987 Workshop on Visual Languages, August, 1987, Linkoping, Sweden

[10] C.J. Date, An Introduction to Database Systems, Volume I, Addison-Wesley Publishing Company, 1981

Overview of Japanese Kanji and Korean Hangul Support on the ADABAS/NATURAL System

Yoshioki Ishii [1] and Hideki Nishimoto [2]

1) Software AG of Far East Inc., JAPAN
2) Ryukoku University, JAPAN

ABSTRACT

This paper describes the progress of Japanese language support at Software AG of Far East and a study on Korean Hangul as well as Japanese Kanji. In addition to, a method called DBCS(double-byte character set) support is introduced. Related problems are discussed and solved by adapting the fourth generation language, NATURAL, as a Software AG product.

1. Japanese Kanji and Korean Hangul

Japanese language borrowed extensively from Chinese Hanzi by way of Korea to form their own written language about 1,500 years ago. Later, the characters came to symbolize native Japanese words similar in meaning to that in Chinese.

Modern Japanese is written as a mixture of ideograms, Kanji, and native phonetic letters, Kana. The Kana phonetic alphabet exists as Hiragana and Katakana, each of which serves different purposes and is stylistically different. A typical piece of Japanese writing contains Kanji, Hiragana, and sometimes Katakana. Each of the Kana syllabaries consists of 46 basic symbols. while Kanji characters are limited to about 2,000 symbols for official and daily use. (Fig.1-1)

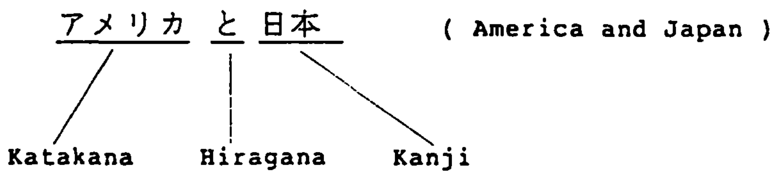
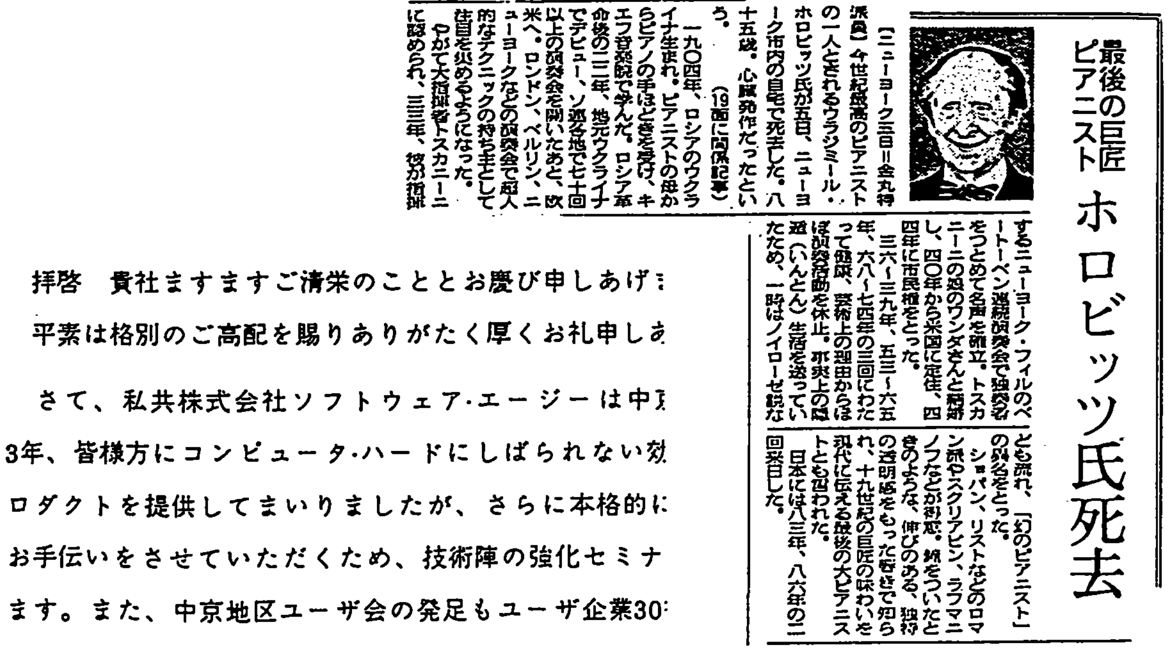


Fig. 1-1 Katakana, Hiragana and Kanji characters

English sentences are horizontally written, from the left to the right. In contrast, Japanese sentences are vertically written, from the top to the bottom. However, it has recently become very popular to write Japanese sentences in the English format. Computer or word processor outputs are usually horizontally written. Figure 1-2 shows a sentence fragment taken from a newspaper article that was vertically written and produced by a word processor as an example of Japanese written in the English format.

When writing vertically, the sentences are formatted from the right to the left of the page. Consequently, the page sequence of Japanese publications that are vertically written is arranged in a different manner than when horizontally written and the publications open from left to right. This also happens in the Chinese language, and is one of the biggest differences between Eastern and Western cultures. Unfortunately, terminals that can display vertically written characters have not yet been developed.


最後の巨匠ピアニスト ホロビッツ氏死去

【ニューヨーク三日＝金丸特派員】今世紀最高のピアニストの一人とされるウラジミール・ホロビッツ氏が五日、ニューヨーク市内の自宅で死去した。八十五歳。心臓発作だったという。（19面に関係記事）

一九〇四年、ロシアのウクライナ生まれ。ピアニストの母からピアノの手ほどきを受け、キエフ音楽院で学んだ。ロシア革命後の二三年、地元ウクライナでデビュー、ソ連各地で七十回以上の演奏会を開いたあと、欧米へ。ロンドン、ベルリン、ニューヨークなどの演奏会で超人的なテクニックの持ち主として注目を集めるようになった。

やがて大指揮者トスカニーニに認められ、三三年、彼が指揮するニューヨーク・フィルのベートーベン連続演奏会で独奏者をつとめて名声を確立。トスカニーニの娘のワンダさんと結婚し、四〇年から米国に定住、四四年に市民権をとった。

三六～三九年、五三～六五年、六八～七四年の三回にわたって健康、芸術上の理由からほぼ演奏活動を休止。私生活上の隠遁（いんとん）生活を送っていたため、一時はノイローゼ説なども流れ、「幻のピアニスト」の別名をとった。

ショパン、リストなどのロマン派やスクリアビン、ラフマニノフなどが得意。鍵をついたときのような、伸びのある、独特の透明感をもった響きで知られ、十九世紀の巨匠の味わいを現代に伝える最後の大ピアニストとも言われた。

日本には八三年、八六年の二回来日した。

拝啓　貴社ますますご清栄のこととお慶び申しあげ

平素は格別のご高配を賜りありがたく厚くお礼申しあ

さて、私共株式会社ソフトウェア·エージーは中

3年、皆様方にコンピュータ·ハードにしばられない

ロダクトを提供してまいりましたが、さらに本格的に

お手伝いをさせていただくため、技術陣の強化セミナ

ます。また、中京地区ユーザ会の発足もユーザ企業30

Fig.1-2 Newspaper Articles and Japanese Wordprocessor Outputs

Japan's computerization started with the use of computers 'made in U.S.A.' At that time, all the printer characters printed were alphanumeric.

A character set consisting of 63 characters was established and Katakana characters were used instead of English, but sentences were still horizontally written. After mainframe computer makers such as IBM released byte addressing machines in 1965, the Roman English alphabet and numerals was added to the Katakana characters and this style continued until about 1980. After 1980, the Japanese characters started being utilized to there fullest extent by computers. Until the 1970's, Japanese characters were only used for names and addresses, but today even text data can be stored in the databases. Computers with Kanji terminals, which represent one character by two bytes and display it on the screen as a Japanese character, have made a lot of progress.

JAPANESE CHARACTER STRINGS	HEX CODE	
日本語漢字 *	0E4562456648E7449A4F5848F20F	←IBM
	28C6FCCBDCB8ECA4CEB4C1BBFA29	←Fujitsu

English translation of * : Japanese Kanji.

Fig.1-3 Japanese character and code representation

In Korean, the use of Hanzi is gradually being phased out, even though the Korean language was also greatly affected by Chinese Hanzi characters.

Now Korean is usually written as a pure phonetic alphabet, called Hangul, with no Hanzi. Hangul was invented in 1446 during the reign of Sejong, the fourth King of the Yi dynasty. It is the only script known to have been designed by a committee.

This phonetic alphabet has 24 letters, consisting of 10 vowels and 14 consonants. The letters are designed in such a way as be grouped into square clusters of two, three, or occasionally four letters. Each cluster represents one spoken syllable of the language. The number of characters required for general use comes to about 2,000 clusters (Fig.1-4).

KOREAN HANGUL LETTERS	DATA
한 글*	<u>0E</u>D0658B69<u>0F</u>

English translation of * : Hangul

Fig.1-4 Korean Hangul and code representation

2. Code Presentation and Typing Method

From early on in the Japanese computer market, there has been an intense interest in establishing a system to handle Kanji. However, it has taken a long time and vast sums of money to create something compatible with the existing one-byte character systems and to develop devices peculiar to Japanese usage (e.g. high resolution displays or graphics printers with special fonts).

Generally one byte is used to denote a character, but a byte with eight bits can provide for as many as 256 characters.

Japanese Kanji must be represented by codes, usually two bytes long, because there are many different kinds of characters(Fig.1-3). This point is pressing computer manufacturers and vendors in Japan to greatly revise the standard systems.

A great number of Kanji typing methods have been developed to date. All of these fall into three main categories; two-dimensional selection arrays, in which the thousands of available characters are laid out as a huge two-dimensional array of keys or pen targets, coding schemes, in which the typist analyzes each character in turn and then figures out or recalls an input code derived from some form of system coding rules, and phonetic conversion, in which the typist spells out the sound of the desired word unit and the computer locates that word in a dictionary stored on one of its disk. In the last category, if several words have the same sound, the typist can indicate the one intended[1]. Currently, phonetic conversion(Fig.2-1) is the most popular method for general users in Japan by reason of its having so few rules to learn and its use of English-typing skills, among other reasons.

```
                     K'ﾆﾎﾝｺﾞ ﾉ ﾃﾞｰﾀ ﾊ ｶﾝｼﾞ ｦ ﾌｸﾑ'


1      ( ﾆﾎﾝｺﾞ                                                    )
 1 日本語

F      ( ﾉ                                                        )
 1 乃  2 廼  3 之  4 埜  5 野  6 脳  7 の  8 ノ  9 嚢

E      ( ﾃﾞｰﾀ                                                     )
 1 データ処理部

F      ( ﾊ                                                        )
 1 羽  2 牙  3 歯  4 刃  5 頗  6 端  7 巴  8 把  9 播 10 覇 11 杷 12 波 13 派
14 琶 15 破 16 簸 17 葉 18 刃 19 玻 20 坪 21 怕 22 霸 23 爬 24 玻 25 皤 26 芭
27 耙 28 菠 29 葩 30 袙 31 跛 32 陂 33 歯 34 羽 35 は

1      ( ｶﾝｼﾞ                                                     )
 1 漢字  2 幹事  3 監事

F      ( ｦ                                                        )


3      ( ﾌｸﾑ                                                      )
 1 含  2 啣  3 含む

                        日本語のデータは漢字を含む *
```

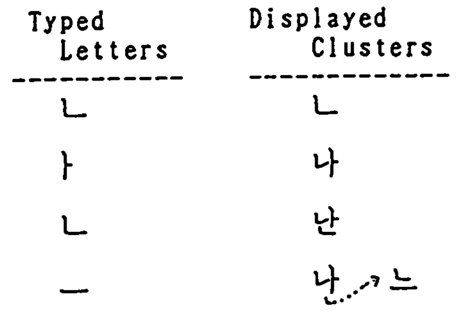
English translation of * : Japanese data includes Kanji characters.

Fig.2-1 An example of the phonetic conversion method

A Hangul cluster may come in different shapes and positions, depending on the surrounding context. A computerized Hangul typing system can have a single generic key or a Romanized phonetic symbol for each Hangul letter. The computer is relied on to decide which is the proper graphic form and position each letter in relation to the letter or letters surrounding it.

Computer displays are very good for "movable" typing because each new letter usually requires revising the form and placement of the letters that have previously been typed[1].

Figure 2-2 represents the typing of a four-letter word. The third letter typed,' ㄴ', has not been fixed at the bottom end of the first character or the top of the second character as it has just been typed. However, the next letter, ' ㅡ', is a vowel, so the system understands that the latter is proper in this case. (Fig.2-2).



Typed Letters	Displayed Clusters
ㄴ	ㄴ
ㅏ	나
ㄴ	난
ㅡ	난 → 느

Fig.2-2 Movable typing of Hangul

3.Product Progress

SAGFE has established several various system environments that are peculiar to Japanese usage since Japanese Kanji was supported on ADABAS* for the first time in 1978.

This support of the Japanese language, which started by putting character strings on a special Kanji printer, has recently been improving with the development of more advanced terminal equipment and controllers.

The history of supporting Kanji on our ADABAS system in Japan can be divided into the following four stages:(Fig.3-1) [2]

Year / Category	1978	79	80	81	82	83	84	85	86	87	88	89
Development of Terminals (Period that users started to use terminals)		Hitachi's T560-20 and Fujitsu's F6650 made it possible to output Kanji characters.			IBM's I3278 made it possible to output Kanji characters.	I5550 was intro-duced.	New emulators for 5550 were introduced.	Kanji characters can be output by many PCs.	----------→			
Release of NATURAL		Release of NATURAL V1.1		Release of NATURAL V1.2						Release of NATURAL 2		
Process of Kanji Support	Output of Kanji characters by batch printers		Output of Kanji characters to terminals by ADASCRIPT		Map functions for rulers and Kanji characters generated in batches		KMAP function			Development of KAPRI 2		
				Output of Kanji characters by NATURAL	Conversion function for Kanji characters and Kana characters		Release of KAPRI				Start of Development of Kanji support by NATURAL itself	

Fig.3-1 The history of Kanji Support in SAGFE

Stage I) Putting Kanji characters on a special printer using batch processing

In 1978, we first supported Japanese Kanji on are law retrieval system developed by an ADABAS system at the Administrative Management Agency(the Management and Coordination Agency at present).

The system is for retrieving law and regulation texts, which the user does by specifying key words. The texts are then printed out on a printer.

Since there was not any terminal equipment for displaying Kanji characters on the market, a special Kanji printer was put on after the retrieval by the batch processing mode.

Stage II) Supporting early Kanji terminals

As soon as early Kanji-supported terminals entered the market in 1980, we tried using the user-friendly query system ADASCRIPT[3] on an ADABAS with the terminal.

In 1981, we succeeded at Kyodo Oil Company in adopting a method by which a map with Kanji characters and ruled lines could be created in advance.

Stage III) Providing a phonetic conversion typing method

In 1982, Software AG succeeded in equipping the fourth generation language NATURAL* system with a phonetic conversion method. The typist first enters the Hiragana spelling of a text directly enclosed with apostrophes, then the word-unit homophones are searched for in the dictionary. After they appear on the screen, the operator can choose the number of the appropriate word. This is the same popular method used for personal computers in Japan. Additionally, our method can be implemented on any non-intelligent terminal because it does not require a special dictionary or an additional processor on the terminal.

Stage IV) Developing the environment for DBCS support

In 1982, an important problem arose when the new version of NATURAL(V1.2) was released since it was equipped with a Map Editor facility for designing the direct output image on the screen.

Processing of the screen I/O is very difficult with our method in this case because the physical length of the data going into the terminal is greater by some bytes for the control-codes than the length of the characters displayed on the screen.

In 1984, IBM developed a virtual terminal emulator for personal computers. This facility adapts the same Kanji shift codes as the Japanese products use. Since then, Kanji character data has been mostly put into the computer through a phonetic conversion processor on an intelligent terminal or on a work-station. Consequently, mainframes mainly undertake data manipulations after the Kanji has been put in.

The first version of NATURAL2 was formally released in 1987. Some of its screen operations have been strongly reinforced. In the same year, SAGFE developed and released the Kanji Proto-typing Interface 2 (KAPRI2)[5] product, which corresponds to the NATURAL2 system. Recently, SAG has decided to design and develop double-byte character set(DBCS) products in order to completely support all the multi-characters sets in the world. There are several manufacturers in Japan, besides IBM that are developing products compatible with IBM software. Products for the operating system, TP monitor, and terminal supported by the ADABAS/NATURAL system are shown in Fig.3-2.

* ADABAS is a DBMS for mainframes developed by Software AG, Germany, in 1971 and is now used at 3,500 sites around the world. NATURAL, a 4th Generation Language also developed by Software AG, is used at 3,300 sites with ADABAS and NATURAL being used together in most cases. At present, ADABAS and NATURAL are run on approximately 500 mainframes in Japan. We had to implement and improve a number of items in order to support the Japanese characters mentioned above.

Manufacturer	OS	TP monitor	Terminal (Note 1)
IBM	MVS, MVS/XA VS1 DOS, DOS/VSE VM/CMS	COM-PLETE TSO CICS IMS	I3270 I3278 I5550 (and other PCs with Japanese 3270 emulators) N6300 (I3270 compatible)
FUJITSU	OSIV/F4 MSP OSIV/X8 FSP	COM-PLETE TSS AIM	F9526 F665X F668X PCs with 6650 emulators (eg. FMR)
HITACHI	VOS 3 VOS 2 VOS 1	TSS ADM DCCMII DCCMIII TMS/3V	H9415 T560-20 PCs with 560-20 emulators (eg. 2020)
MITSUBISHI	GOS/VS	TSS CIMS	M4378 PCs with emulators

Note 1:
If a terminal can be connectd to other manufacturers' OSs and TP monitors, then the terminal is able to work with those OSs and TP monitors, i.e. terminals can be supported independent of OSs and TP monitors.

Fig.3-2 ADABAS/NATURAL environments supported in Japan

4.DBCS Support of the Approach

Problems common to the support of handling multi-characters are discussed in this section. The first problem is that several code notations representing the same character set already exist. In Japan, manufacturers naturally established original code specifications before the computerized code notation of each character became standardized within the country. The two types of code tables for Japanese Kanji are shown in Fig.4-1.

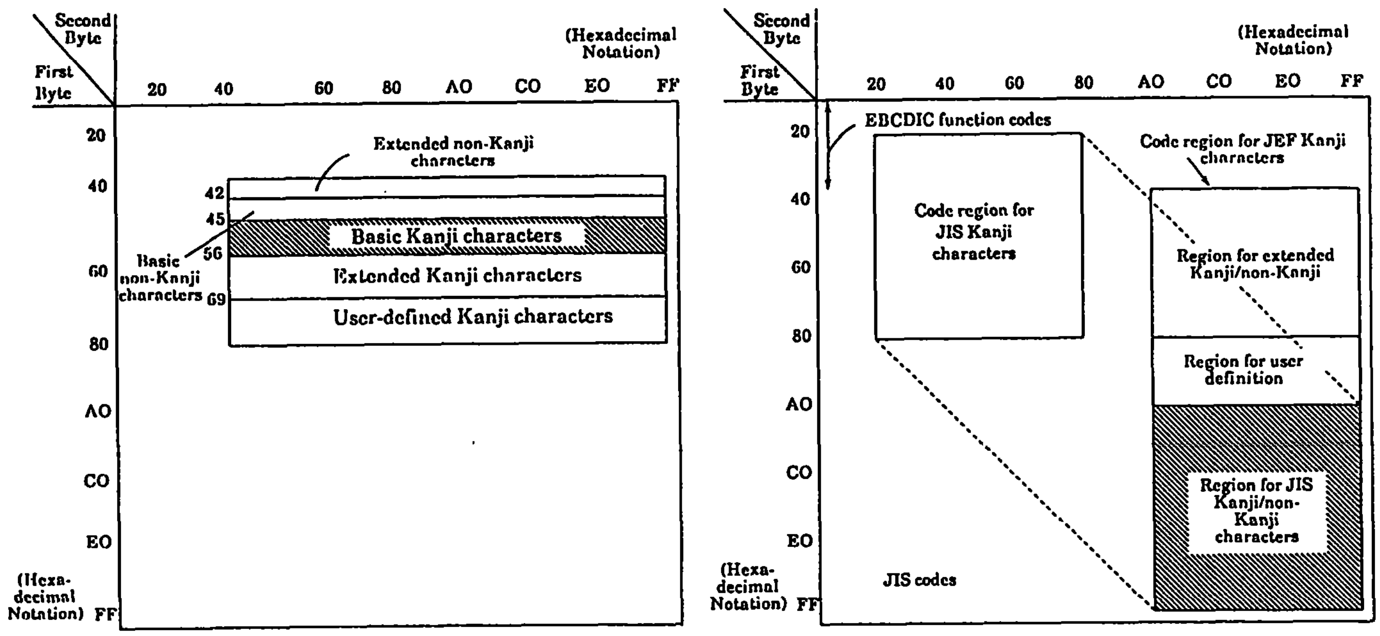

Second Byte
First Byte
(Hexadecimal Notation)
20 40 60 80 AO CO EO FF
20 40 60 80 AO CO EO
(Hexa-decimal Notation) FF
Extended non-Kanji characters
42
45
56
69
Basic non-Kanji characters
Basic Kanji characters
Extended Kanji characters
User-defined Kanji characters
Second Byte
First Byte
(Hexadecimal Notation)
20 40 60 80 AO CO EO FF
20 40 60 80 AO CO EO
(Hexa-decimal Notation) FF
EBCDIC function codes
Code region for JIS Kanji characters
Code region for JEF Kanji characters
Region for extended Kanji/non-Kanji
Region for user definition
Region for JIS Kanji/non-Kanji characters
JIS codes

Fig.4-1 IBM and JIS Kanji codes

Although the IBM Kanji code is organized as an expansion of EBCDIC, the JEF code by Fujitsu and the KEIS code by Hitachi adopt the shifted JIS code. In a distributed processing environment, character strings that belong to the same character sets, even if based on different code notations, must be correctly executed in comparison or move operations. Consequently, a data transformation function may be additionally required, since this is seen as a very important role of the database management system.

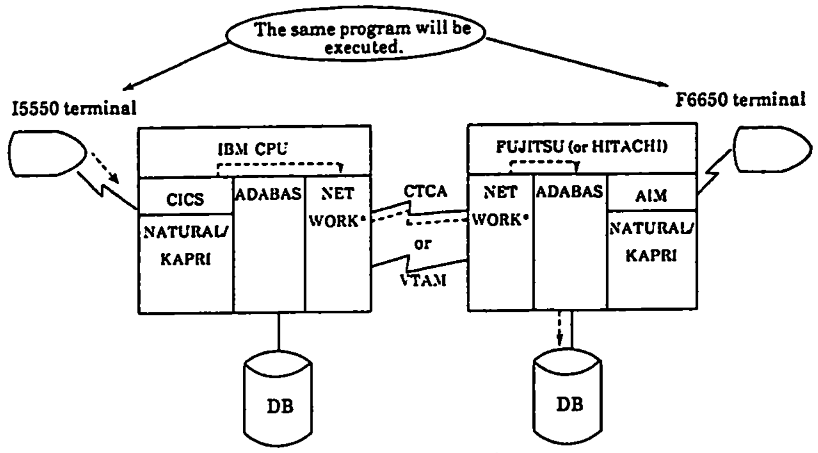


Fig.4-2 ADABAS/NATURAL distributed environment

Second, there may be many difficulties in the handling of the shift control codes. In a terminal data stream, double-byte characters are usually distinguished from single-byte characters by checking the beginning and end of strings with a Shift Out/Shift In(SO/SI) control code.

Each manufacturer has its own specifications for these codes, and this is a serious problem in a distributed processing environment.

A database management system should also be able to completely transform the control codes used at any site(Fig.4-2). Additionally, the internal procedures for data manipulation - Search, Move, Edit and others - should be revised using various options to keep the results consistent(Fig.4-3).

```
0010 **
0020 ** NON SUPPORTED DBCS FACILITY PROGRAM SAMPLE
0030 **
0040 ** MOVE & COMPRESS
0050 MOVE ' 日本語の文章 ' TO #WORK1(A14)
0060 MOVE #WORK1 TO #WORK2(A6)
0070 COMPRESS #WORK2 'JAPAN' TO #WORK3(A14) LEAVING NO SPACE
0080 DISPLAY #WORK2 / #WORK 2(EM = H(6)) #WORK3 / #WORK3(EM = H(14))
0090 END
```

```
        #WORK2                    #WORK3

 ------------------  ---------------------

  日本                      日本[illegible]■■               →character representation
 0E4562456648          0E4562456648D1C1D7C1D5404040        →HEX notation

      (after MOVE)               (after COMPRESS)
```

Some data may be corrupted when an SI code is truncated after executing the MOVE statement and the SI code for DBCS data can not be specified.

Fig.4-3 Example of data manipulation with trouble

The double byte character set (DBCS) support [4] of ADABAS/NATURAL consists of two products, the ADABAS/NATURAL code standardization and the DBCS facility.

The first product is designed to correctly process data manipulations, even among different code notations belonging to the same character set.

In the ADABAS/NATURAL world, all representations of character strings will usually conform to a standard code notation that has been established for each character set in advance (Fig.4-4). DBMS should keep track of the environments of every site and give a proper code transformation during I/O communication only. Consequently, the internal procedures of the data manipulation can keep the results consistent and each site does not need any additional transformation programs or huge tables.

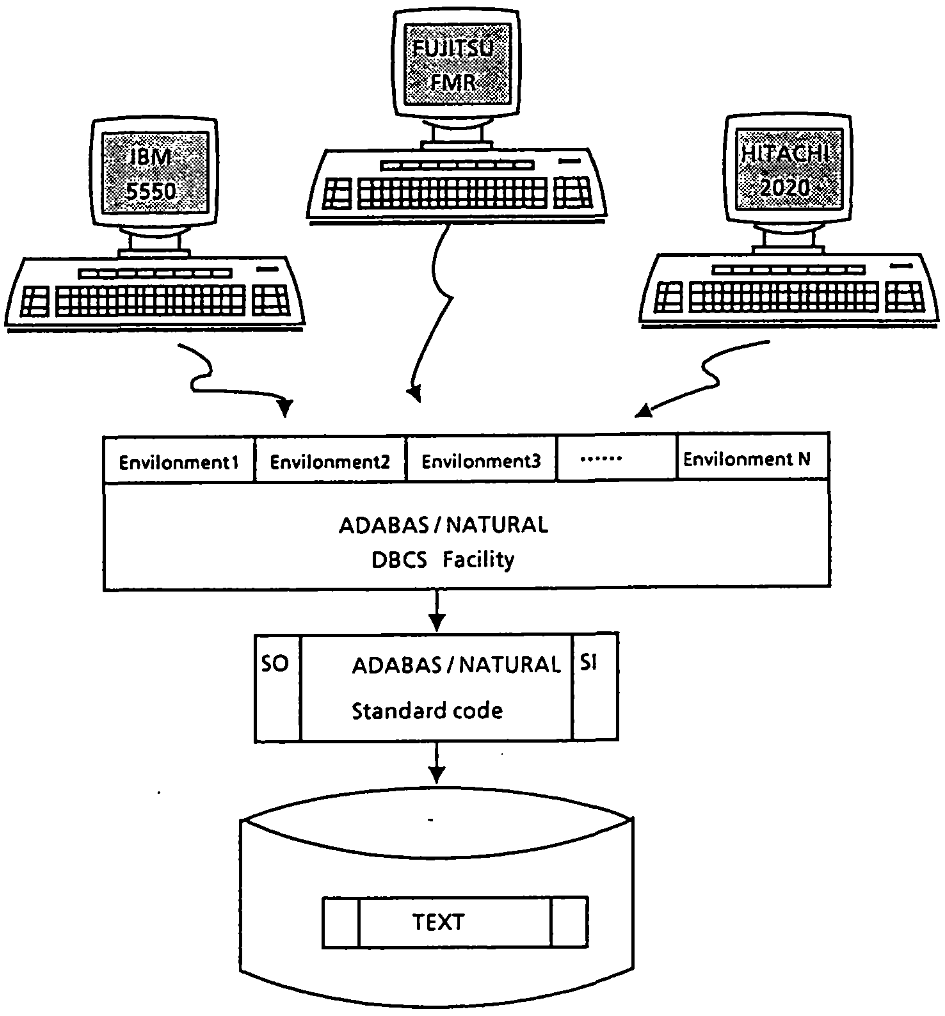
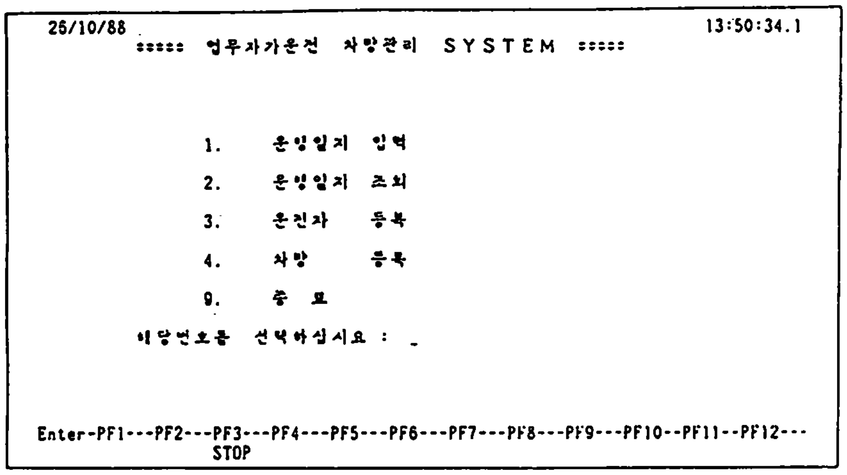


Fig.4-4 ADABAS/NATURAL standard codes

The second product, the DBCS facility, has been developed so that all NATURAL operations can completely accept every variable, name, and data value described by any double-byte character as well as any single-byte character defined as EBCDIC or ASCII (Fig.4-5).



```
25/10/88                                                           13:50:34.1
        =====  업무자가운전  차량관리  S Y S T E M  =====


                  1.     운행일지  입력
                  2.     운행일지  조회
                  3.     운전자    등록
                  4.     차량      등록
                  9.     종  료
        해당번호를  선택하십시요 :  _


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
                  STOP
```

English translation of the menu written in Hangul:

Staff driving car management system
1. Daily driving input
2. Daily driving inquiry
3. Driver registration
4. Car registration
9. Exit
Please select number : _

Fig. 4-5 Screen supported DBCS

5.Conclusion

We plan to design a complete DBCS support facility and build it into the nucleus of the ADABAS/NATURAL system in order to correctly deal with double-byte characters for various data manipulations. This may make it easier to build applications for handling other multi-characters sets, in addition to Kanji or Hangul, if the character data is managed on the ADABAS/NATURAL standard code system and performed using the DBCS facility we have presented.

REFERENCES

[1] J.D.Becker,"Typing Chinese, Japanese, and Korea", COMPUTER, Jan.1985,pp.27-34.
[2] M.Shimada et al., "Supporting the Character Sets of Japanese Kanji and Korean Hangul in the ADABAS/NATURAL System", Proc. of Int. Symp. on Database Systems for Advanced Applications, 1989, pp.3-9.
[3] P.Schnell,"Implementation and Engineering of a Production-oriented DBMS", INFORMATION PROCESSING 83, Elsevier Science Publishers B.V.,1983,pp.637-646.
[4] Software AG, "Double Byte Character Set", Software AG Internal Document,1988
[5] Software AG of Far East, "KAPRI Reference Manual"(In Japanese), Tokyo,1988
[6] Y. Ishii, "Introducing ADABAS to Jananese Market", Database Engineering Mar. 1983.
[7] J. Martin, "4GL Fourth Generation Languages", Prentice Hall, 1986.

9th International Conference on Entity-Relationship Approach

October 8-10, 1990, Lausanne, Switzerland

Conference Chairman	:	**Dennis Tsichritzis**, University of Geneva
American Conference Co-Chair	:	**Kathi Hogshead Davis**, University of Northern Illinois
Program Committee Chairman	:	**Hannu Kangassalo**, University of Tampere
Steering Committee Vice-Chair	:	**Sal March**, University of Minnesota
Local Organization	:	Database Laboratory, Swiss Federal Institute of Technology

The Entity-Relationship Approach is used in many data base and information systems design methodologies and tools. Its use is expanding to new types of applications and the ER model itself is being developed to meet new requirements posed on the advanced modelling approach. This ER conference continues its tradition of bringing together practitioners and researchers to share ideas, experiences and new developments and issues related to the use of the ER Approach. The conference consists of presented papers, both on the theory and practice of the ER Approach, as well as invited papers, tutorials, panel sessions, and demonstrations.

Major Topics :

Knowledge Representation

- Models for knowledge representation
- Schema evolution and maintenance
- Schema as a knowledge base
- Model taxonomies

Conceptual Modelling and Data Base Design

- Acquisition of modelling knowledge
- Support systems for conceptual modelling and knowledge engineering
- Design methodologies,tools and interfaces
- Practical experiences

New Approaches in Database Management Systems and in Information Systems

- Object-oriented systems
- Schema and query translation
- User interfaces, visual query langages
- Distributed ER schema
- Multimedia database systems
- Special-purpose systems
- Practical experiences

Innovative theories and applications

- Accounting
- Linguistics
- Social sciences
- Software engineering
- Music composition
- Urban data management

Submission of Papers :

Original papers are sollicited on the above topics or any other topic relevant to the ER approach. Five copies of completed papers must be received by the program committee chairman by April 2, 1990. Papers must be written in English and may not exceed 25 double spaced pages. Selected papers will be published in *Data and Knowledge Engineering* (North-Holland).

Important Dates :			
	Papers due	:	April 2, 1990
	Notification of acceptance	:	June 10, 1990
	Camera ready copies due	:	August 6, 1990

All requests and submissions related to the program should be sent to the program committee chairman :

Hannu Kangassalo
University of Tampere
Department of Computer Science
P.O. Box 607
SF-33101 Tampere, Finland

tel : +358-31-156778
fax : +358-31-134478
e-mail : hk@utacs.uta.fi

CALL FOR PAPERS

International Conference on Multimedia Information Systems

January 16 - 18, 1991

Co-Organizers : **ACM SIGIR and The Institute of Systems Science National University of Singapore**

In Co-operation with : ACM SIGOIS **Co-operation sought from :** ACM SIGGRAPH, ACM SIGCHI and ACM SIGMOD

Theme: MULTIMEDIA INFORMATION SYSTEMS - TOWARDS BETTER INTEGRATION

The last few years have seen some dramatic trends in computer technology: Faster workstations, higher resolution displays, cheaper and larger storage, improvements in graphics packages and progress in broad-band communications. While all these topics have forums where the research, development and application efforts are reported, there is yet to be a forum which promotes discussion and exchange of ideas on how all these interesting technologies could converge towards the next generation of tools and applications. This effort is very timely since one perceives a confluence of three major industrial sectors: Publishing and printing, Video and Movie, and Computer technology. Through this conference, we hope to bring together people from various individual technologies to exploit the combined advantages in building integrated systems using some or all of the technologies.

Papers

Technical papers reporting either original research or interesting applications are invited in areas relevant to the theme. Topics of interest include but are not limited to the following:
Hardware: Optical Disk devices and their use, High Bandwidth Buses, Multimedia networking, Communication protocols for Multimedia systems, etc.
Systems Software: Distributed Multimedia architectures, Support for Multimedia Databases, Object-oriented Multimedia systems, Co-operative authoring tools for Multimedia Systems, Video and Still Image compression, etc.
Content-based retrieval of Multimedia Objects: Object oriented languages for multimedia retrieval, Retrieval in hypertext/hypermedia networks, Hierarchical languages, Access methods for multimedia systems, Indexes for video, images and voice, Browsing in very large images and video, etc.
User Interfaces: Visual interfaces to Multimedia systems, Browsing interfaces, Agents in interfaces, Physical metaphors, Adaptive interfaces, Authoring and editing tools.
Applications and Systems: Picture and document-image archival systems, Standards for electronic document preparation and submission, Engineering applications, Medical Information Systems, Libraries of the future, Multimedia in education, Home entertainment systems.

Panels

Proposals are invited for panels relevant to the theme of the conference. The person proposing a panel should identify the topic and the participants. The co-ordinator should also identify alternates. Written consent is required from the proposed panelists agreeing to participate in the panel. Proposals for panels should be received by 31 March, 1990

Tutorials

Proposals for tutorials are also welcomed on topics relevant to the theme of the conference. The person proposing the tutorial should give an outline of the topic and indicate the duration of the tutorial. Proposals for tutorials should be sent in by 31 March, 1990.

Demonstrations

Proposals are sought for demonstrations during the conference. These can be either stand alone or on-line connected to overseas facilities. Proposals should be sent by July 1, 1990.

Special Sessions

Proposals are invited for organizing special sessions which focus on emerging technologies. Proposals for special sessions should be received by May 1, 1990.

Deadlines

Four copies of full papers should be (20 pages or approximately 500 words) sent to the following addresses **by 10 April, 1990.**

North America and Europe:
Professor Stavros Christodoulakis
Institute for Computer Research
University of Waterloo
Waterloo, Ontario, Canada N2L 3 G1
Bitnet: Watmath!WatDaisy!Schrisodoul @ uunet.uu.net
FAX: 1-519-8851208

Australia, Asia and Others:
Dr Desai Narasimhalu
Institute of Systems Science
National University of Singapore
Heng Mui Keng Terrace
Singapore 0511
Bitnet: ISSAD @ NUSVM
FAX : 65-7782571

- Notification of acceptance of papers will be mailed to authors on or before 1 July, 1990.
- Accepted papers, camera ready, are due no later than 1 September, 1990.

IEEE Computer Society
1730 Massachusetts Avenue, N W
Washington, DC 20036-1903

Non-profit Org
U.S. Postage
PAID
Silver Spring, MD
Permit 1398

Henry F. Korth
University of Texas
Taylor 2124 Dept of CS
Austin, TX 78712
USA