Mule: MULtilingual Enhancement to GNU Emacs

錦見 美貴子[1]
半田 剣一[2]
戸村 哲[3]

Abstract

Mule: MULtilingual Enhancement to GNU Emacs handles not only ASCII characters (7 bits) and ISO Latin-1 (8 bits), but also Greek, Cyrillic, Hebrew, Thai, Vietnamese (8 bits), Japanese, Chinese, Korean (16 bits) coded in the ISO2022 standard and its variants (e.g. EUC, Compound Text).

A text buffer of Mule holds a mixture of these languages. As for inputting, Mule provides various input methods and in case Mule is running under a terminal emulator, whatever the emulator provides can be used.

Mule has extended GNU Emacs's internal structure on many points including buffer, string, character and character-syntax so that EmacsLisp can freely manipulate multilingual text. In addition, Mule converts multilingual text into appropriate codes while handling many types of I/O (e.g. file, process, keyboard, display).

Since Mule is upper compatible with the original GNU Emacs, there's no regression for users of the original even if they move to Mule.

I. Introduction

In this world where we electronically create, exchange and consume text of many languages, a multilingual environment is a fundamental of text processing systems. A multi-lingual environment is not a medley of multiple single-lingual environments. Characters or texts of many languages may appear in one buffer and each language fragment requires special treatment to satisfy its orthography. Characters of different languages may have different size or coding and texts of different languages may have different writing direction or word definition. A multi-lingual environment enables users to switch around different languages.

In the design of such an environment, we have to give extra consideration on two points: one concerning about handling of multiple character sets and coding systems and the other about realization of environment for individual languages. We have augmented GNU Emacs[2], one of the most popular text editors, in these two points and realized Mule, a MULtilingual Enhancement to GNU Emacs.

Mule is practically upper-compatible to original GNU Emacs and

- handles 8 bit and 16 bit character sets. Some character sets are predefined and users can add new ones. The representation of character sets is efficient and each character sets can be easily distinguished from other sets.
- proposes a classification for coding systems. Users can define new coding system according to Mule's classification.
- provides character input methods through ASCII keyboards.
- enables users to define language-specific syntax about characters and words.

In this paper, we first show how Mule handles multiple coding systems and character sets and then explain about input systems for individual languages. We then describe how users can customize the environment and other changes from original GNU Emacs.

II. Multiple Character sets and Coding systems

Figure 1 shows a snapshot of Mule system. The buffer contains ASCII characters, Japanese characters and many other characters from different character sets. A multi-lingual system has to represent 8-bit character sets and 16-bit ones in an integrated manner and each character sets must be easily distinguishable. Another point to be considered is coding systems. Many coding systems exist in the world and they should be systematically handled. Furthermore, character sets and coding systems are constantly increasing, so multi-lingual system has to support plain and easy way to define new ones.

[1]Nishikimi Mikiko. Electrotechnical Lab. `<nisikimi@etl.go.jp>`
[2]Handa Kenichi. Electrotechnical Lab. `<handa@etl.go.jp>`
[3]Tomura Satoru. Electrotechnical Lab. `<tomura@etl.go.jp>`

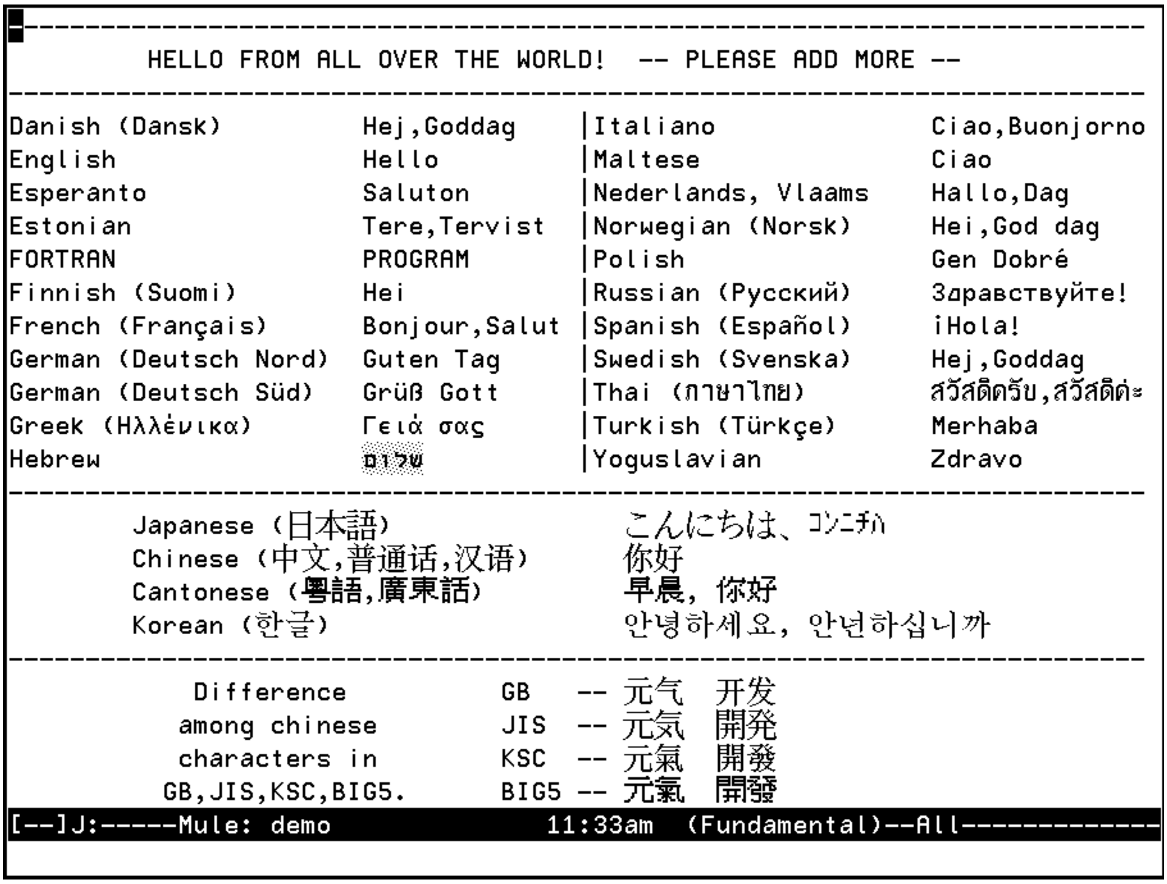

Figure 1: A snapshot of Mule window

II.A. Character sets

Mule can handle every character set that meets technical requirements standardized in ISO 2022[4]. Among character sets registered in ECMA[5], Mule supports those shown in Table 1 as default.

Mule can handle private character sets also. They can be defined by setting its seven attributes: byte length of code, columns occupied on a screen, the type of the character set(94-char-set, 96-char-set, 94x94-char-set, or 96x96 char-set), the graphic set (GL or GR), the final character ('0' through '?'), the displaying direction (Left-to-right or Right-to-left) and *leading character* described in subsection III.A. Two Chinese private character sets (Big5 and PinYin-ZhuYin) are predefined and users can add as many sets as needed.

In ISO 2022, representing a character requires two phases, first designate one character set to one of the four registers (G0,G1,G2,G3), and then invoke invoke the register that contains the character. ISO 2022 prohibits designating 96-character sets to G0 from historical reason, but for the simplicity Mule supports this designation and we have defined intermediate character `,` `[0x2C]` to designate a 96-character set to G0.

One more deviation from ISO 2022 is that Mule supports Big5, a Chinese character sets, which does not meet its technical requirement[6].

II.B. Coding System

One character sets can be encoded in many coding systems. Different coding systems are used in different OS or different applications. Mule supports MS Kanji Code (Shift-JIS), Big5 Chinese character code, and all ISO 2022 variants.

There are many ways of encoding multilingual text along ISO 2022. So called JIS-code (Japanese JUNET code), EUC[Extended UNIX Code], and Compound Text of X are all its variants. Mule classifies ISO 2022 variants by the following attributes.

[4]**The International Register** contains character sets which have been registered along ISO 2375. Any character set can be a candidate for registration so long as it satisfies the technical requirements fo ISO 2022 and the format requirements of ISO 2375.

[5]ECMA (European Computer Manufacturers Association) is the Registrations Authority for ISO 2375.

[6]As Big5 contains more than 13,000 characters, Mule cannot handle Big5 as one character set and thus we divide this set into two subsets internally.

Table 1: Mule's default character sets

TYPE	NAME	<F>
94 character set	ISO 646 USA (ASCII)	4/2
	JIS X 0201 Japanese Kana	4/9
	JIS X 0201 Japanese Roman	4/10
96 character set	Right half of ISO 8859-1 Latin alphabet No.1	4/1
	Right half of ISO 8859-2 Latin alphabet No.2	4/2
	Right half of ISO 8859-3 Latin alphabet No.3	4/3
	Right half of ISO 8859-4 Latin alphabet No.4	4/4
	Right half of ISO 8859-7 Greek alphabet	4/6
	Right half of ISO 8859-6 Arabic alphabet	4/7
	Right half of ISO 8859-8 Hebrew alphabet	4/8
	Right half of ISO 8859-5 Cyrillic alphabet	4/12
	Right half of ISO 8859-9 Latin alphabet No.5	4/13
	TIS 620-2533 Thai Character Set	5/4
94x94 character set	JIS C 6226-1978 Japanese	4/0
	GB 2312-1980 Chinese	4/1
	JIS X 0208-1983 Japanese	4/2
	KS C 5601-1987 Korean	4/3
	JIS X 0212 Japanese Supplement	4/4
	CNS 11643 Set 1 Chinese	4/7
	CNS 11643 Set 2 Chinese	4/8

Table 2: Mule's default ISO 2022 variant coding systems and their characterization

Code	1				2	3	4	5	6
	G0	G1	G2	G3					
Japanese network	ASCII	not used	not used	not used	Yes	Yes	Yes	7	No
Japanese EUC	ASCII	JISX0208-1983	JISX0201-Kana	JISX0212	Yes	Yes	Yes	8	No
Chinese EUC	ASCII	GB2312	not used	not used	Yes	Yes	Yes	8	No
Korean EUC	ASCII	KSC5601	not used	not used	Yes	Yes	Yes	8	No
Korean network	ASCII	KSC5601	not use	not used	No	No	Yes	7	Yes
Compound Text	ASCII	Latin-1	not used	not used	No	No	Yes	8	No

1. What are the initial designation to G0 through G3?
2. Does the code allow short form designation for Japanese and Chinese?
3. Must ASCII be designated to G0 before control characters?
4. Must ASCII be designated to G0 at the end of line?
5. Is the code defined in 7-bit environment or 8-bit environment?
6. Does the code use locking Shift?

Mule has, as default, ISO 2022 variant coding systems shown in Table 2 and users can define any ISO 2022 variants by specifying these attributes.

III. Internal representation

In order to represent characters of many character sets described in subsection II.A, Mule utilizes its own internal code.

Mule classifies characters (thus character sets) by numbers of bytes they are originally coded in (the former digit in the type name in the list below), and numbers of bytes they are internally represented in (the latter digit). The classes are:

Type 1-1 ASCII characters

Type 1-2 Characters in one-byte character sets (e.g. ISO 8859-1 Latin-1)

Type 1-3 Characters in one-byte character sets of private use

Type 2-3 Characters in two-byte character sets (e.g. JIS X0208 Japanese)

Table 3: Extended Character Object

type	value of character object	range
Type 1-1	`C`	`0...0x7F`
Type 1-2	$((\texttt{LC1}\ \&\texttt{0x7F}) \ll \texttt{16}) \mid (\texttt{C1} \ll \texttt{8})$	`0x01A000...0x0FFF00`
Type 1-3	$((\texttt{LC21}\ \&\texttt{0x7F}) \ll \texttt{16}) \mid (\texttt{C1} \ll \texttt{8})$	`0x20A000...0x3FFF00`
Type 2-3	$((\texttt{LC2}\ \&\texttt{0x7F}) \ll \texttt{16}) \mid (\texttt{C21} \ll \texttt{8}) \mid \texttt{C22}$	`0x10A0A0...0x19FFFF`
Type 2-3	$((\texttt{LC22}\ \&\texttt{0x7F}) \ll \texttt{16}) \mid (\texttt{C21} \ll \texttt{8}) \mid \texttt{C22}$	`0x40A0A0...0x5FFFFF`

Type 2-4 Characters in two-byte character sets of private use

Type N Composite characters of variable length

III.A. Buffer and String

In a buffer or in a string, characters of each class has its own representations. Except for ASCII characters which are represented as they are, every character is encoded in the range 0xA0 ∼ 0xFF and added one *leading character* at the head to identify its character sets. Every leading character is coded in one-byte of the range 0x80 ∼ 0x99 (for official character sets) and of the range 0xA0 ∼ 0xFF (for private character sets). In a buffer or a string, however, a leading character for a private character set is preceded by one extra byte (0x9A ∼ 0x9D) to distingush them from a character code. This extra byte also serves as a indicater of byte-length and column-width for the character.

The following list shows the internal representations for a character of each of the classes. In the list, `Cmn` means [`0x80` | `n`-th byte of the `m`-byte character in its original coding] and ranges from `0xA0` to `0xFF`. For example, `C21` is the result of the `OR` operation of `0x80` and the first byte for a character coded in 2-byte.

Type 1-1 `C`
one-byte original code for the character `C` (`C` ≤ `0x7F`), that is, represented as it is.

Type 1-2 `LC12 C11`
sequence of a one-byte leading character and a one-byte original code for the character.
`0x81` ≤ `LC12` ≤ `0x8F`

Type 1-3 `LCPrv1 LC13 C1`
sequence of a two-byte leading character and a one-byte original code for the character,
where for character of one-column width:
`LCPrv1` = `0x9A`, `0xA0` ≤ `LC13` ≤ `0xB7`
for character of two-column width:
`LCPrv1` = `0x9B`, `0xB8` ≤ `LC13` ≤ `0xBF`

Type 2-3 `LC23 C21 C22`
sequence of a one-byte leading character and a two-byte original code for the character.
`0x90` ≤ `LC23` ≤ `0x99`

Type 2-4 `LCPrv2 LC24 C21 C22`
sequence of a two-byte leading character and a two-byte original code for the character,
where for character of one-column width:
`LCPrv2` = `0x9C`, `0xC0` ≤ `LC24` ≤ `0xC7`
for character of two-column width:
`LCPrv2` = `0x9D`, `0xC8` ≤ `LC24` ≤ `0xDF`

Type N `LCCMP LCN1... LCN2... LCNn...`
sequence of a one-byte leading character indicating a composite character and n-byte sequence of element characters (each element has its leading character).
`LCCMP` = `0x80`
`LCNn` = `0xA0` [for ASCII]
`LCNn` = `LCmn` + `0x20` [for others]

For instance, the internal byte sequence for the "LATIN_CAPITAL_LETTER_A_GRAVE" is:
`0x81 0xC0`
because the type is 1-2, the code of the character in ISO8859-1 is `0xC0`, and the leading character for ISO8859-1 is `0x81`.

III.B. Character object in EmacsLisp

EmacsLisp treats character objects as integers of values less than 256. Mule recognizes much more character objects and we have extended character objects. Each character set class except Type N has its own representation. Type N characters cannot be treated as a character object. Table 3 shows character objects of each class. For instance, the internal character code for the Type 1-2 character "LATIN_CAPITAL_LETTER_A_GRAVE" is:
$\texttt{81920} = ((\texttt{0x81}\&\texttt{0x7F}) \ll \texttt{16}) \mid (\texttt{0xC0} \ll \texttt{8})$.

IV. Inputting method

In most cases ASCII keyboards are the only means to input multi-lingual text. There is no key that has a Chinese character or a Hangul on its top. Mule supports four kinds of inputting methods: two for various languages and two for Japanese only.

```
#### Demonstration of Quail's Latin package ####

Keystrokes:     Gra^ce a` vous, tout s'est bien passe'.
Interted text:  Grâce à vous, tout s'est bien passé.

Keystrokes:     Ich bin der U"berzeugung, das/ er heute kommt.
Interted Text:  Ich bin def Überzeugung, daß er heute kommt.

Keystrokes:     !/Hora! ?/Co'mo esta's?
Interted Text:  ¡Hola! ¿Cómo estás?

[--]J:-----Mule: quail_tamago       1:07pm  (Text Q-LATIN-1 Fill)--Top-
```

Figure 2: Inputting latin characters

```
#### Demonstration of Quail's Hangul (한글 2벌식) package ####

  Typing "gks" changes buffer as follows:
        g       gk      gks
        ㅎ      하      한
  Typing "rkrk" changes buffer as follows:
        r       rk      rkr     rkrk
        ㄱ      가      각      각ㅏ
  Typing "rk<SP>rk" changes buffer as follows:
        r       rk      rk<SP>r rk<SP>rk
        ㄱ      가      가ㄱ    가가

[--]J:--**-Mule: quail              10:43am  (Text Q-한글 2벌식 Fill)--
```

Figure 3: Translation using a context dependent rule in Quail

IV.A. Quail

Quail is a simple key-translation system which enables users to input multilingual text from normal ASCII keyboard. We provide quail-packages for Chinese [GB and Big5] (using PinYin, PinYin with Tone, CCDOSPY, SW, ETZY and ZOZY), Japanese (using TCode), Cyrillic, Greek, Hangul, Hebrew, Latin (French, German, Swedish etc.), Thai and Vietnamise. A quail-package contains a translation table for one of the character sets.

Figure 2 shows Latin input through Quail. When you enter in quail-mode, the current package is shown in the modeline (`Q-LATIN-1` in this example). Users can load several packages and select one as the current package. In the example, lines beginning with `Keystrokes:` displays keys user typed in and the lines below them (beginning with `Inserted Text:`) contains characters they are translated into.

Translation rules in a quail-package are not always unique. Rules may be ambiguous or context dependent. If there are more than one candidate translation for the ASCII sequence typed in, the first candidate is shown in buffer with underscore. Users can look another candidate one by one for the intended translation, or browse the list of all candidates and select one. If there is only one possible translation, underscore will not appear.

Translation rules may, moreover, be context dependent. Quail-packages can be divided into two types: those that contain context dependent rules and those that do not.

Context dependent packages include rules like $ab \rightarrow A, aba \rightarrow B, b \rightarrow C$ and as Quail does leftmost derivation and searches longest match, typing $abab$ will enter BC. In order to input AA, the input sequence must be explicitly delimited after first ab. Hitting a space key delimits input and Quail forcibly translates the current input sequence.

Figure 3 show translations using ambiguous and

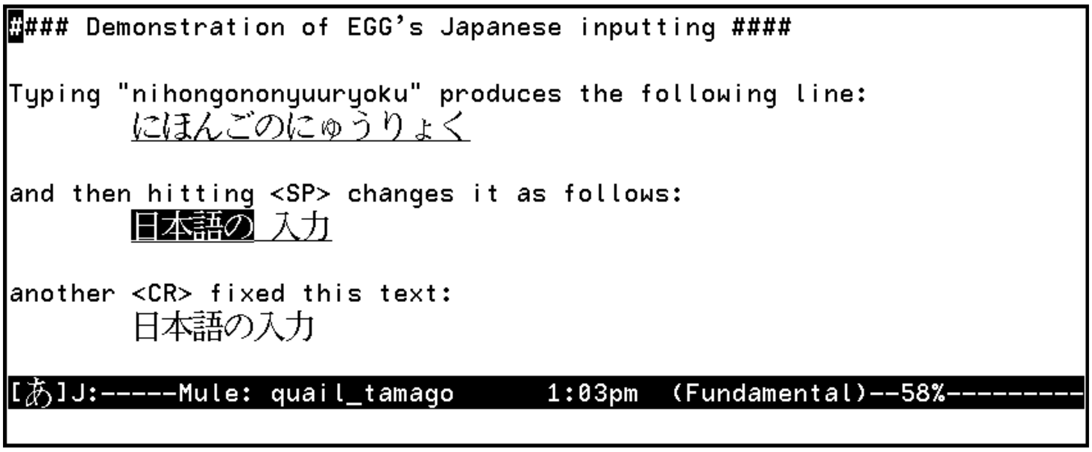

Figure 4: Kanji Input through Tamago

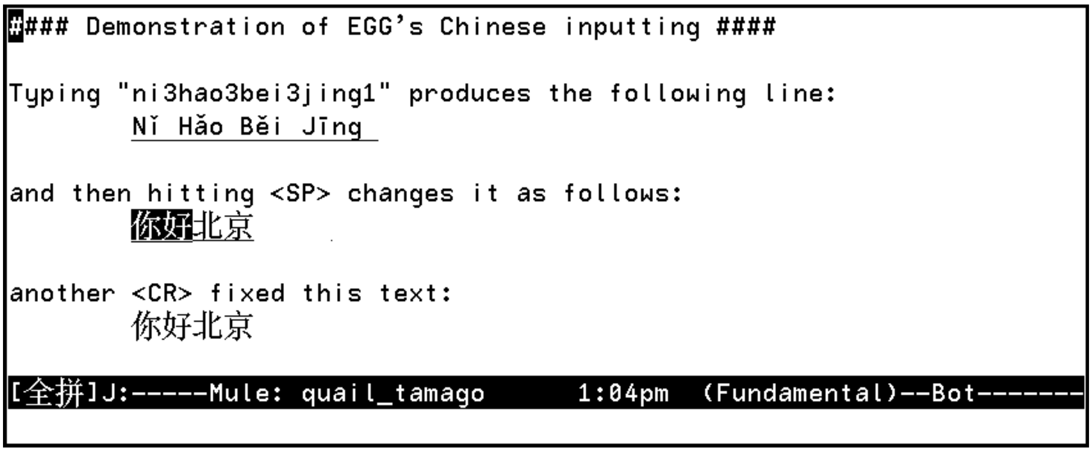

Figure 5: Chinese character Input through Tamago

context dependent rules. In the first example, every sequence `g`, `gk` and `gks` has its own translation and as `g` and `gk` do not determine one translation, underscores appear with their candidate translations. The second and the third examples display the difference between `rkrk` and `rk<SPC>rk`.

While using context independent packages, there is no need for delimiters. Quail sticks to longest match and users need extra key strokes only when they want to choose a translation that is not a first candidate.

IV.B. Tamago-ver.Takana

Tamago-ver.Takana is a character input system that requires a Kana-Kanji conversion server (jserver/Wnn[1]) on the network to input Japanese and a PinYin-Chinese character conversion server[3] for Chinese. Tamago has translation tables from keyboard input to Kana or PinYin. These tables are called input-translation-systems. After input is translated into Kana or PinYin, Tamago communicates with conversion servers to make them into Kanji or Chinese characters. Tamago has also a translation table for Hangul, thus Hangul-Hanzi mixture can be input with a Hangul-Hanzi server[7].

Figure 4 shows how Tamago converts normal keyboard input into Japanese characters. Text that appears in the brackets at the left of modeline shows the current input-translation system. With this translation table, Latin alphabets that are used to represent Romaji are translated into Kana and symbols are translated into Japanese symbols (mostly double width version of the original symbols).

When the readings of the text have been successfully input, hitting the space triggers dividing text into lexical items and conversion of each items into Kana-Kanji mixtures. Users can change the length of the items, try another Kanji conversion of the item at the echo area or browse all the candidate Kanji.

Chinese character input is a similar process except

[7] Wnn consortium is now planning a Hangul-Hanzi dictionary.

PinYin is used instead of Kana and no PinYin is left when conversion is finished. Figure 5 demonstrates Chinese character input using Tamago. Tamago divides PinYin sequence into readings of one character and convert each reading into one character.

IV.C. Other Japanese inputting methods

Other Japanese input systems Mule supports are:

SJ3-egg
: SJ3-tamago has the same user interface as Tamago, but communicates with a Kana-Kanji conversion server sj3serv/SJ3 instead of jserver to convert Kana into Kanji.

Canna
: Canna provides the same user interface not only for Emacs or Mule, but also for X environment (kinput2) and the other TTY environment (canuum). Cannaserver does the Kana-Kanji conversion.

V. Customizing for individual languages

Each language may have its own character set, coding system, input system and word syntax. Mule enables users to build up environments of their own.

Users can define new character set by a EmacsLisp function `new-character-set` that specifies character set attributes described in subsection II.A. Another EmacsLisp function `make-coding-system` defines ISO 2022 variant coding systems.

Quail input system is also customizable. Users may define new Quail packages and realize arbitrary translation. A Quail package is a library that includes three strings (name, prompt in the modeline, document), two flags (one to control its learning of the priority of translation candidates and one to specify whether rules in the package is context dependent or not), a keymap that is used in Quail mode and two association lists. One of the association lists define translation rules and the other specify what string is shown in echo area when a certain character is typed in.

Concept of character and word differs among languages. For example, in Japanese most characters are encoded in two bytes and *bunsetu* (a word and one or more particles) is often used in replacing or searching a text fragment.

We have extended character syntax of GNU Emacs for multi-byte characters. In addition, we have introduced *character category*. Character categories is like character syntax, except that each character may belong to more than one categories and users can define new category types. Predefined character categories are shown in Table 4. Regular expression handler are also extended to properly handle character syntax and category. For instance, emacslisp program:

```
(re-search-forward "\\ch")
```

searches for Hangul characters in a buffer.

A word is also definable in Mule, while original GNU Emacs restricts a word to a sequence of word constituents. Users can define as *word* any word-like part of text using regular expression and word by word operation such as `forward-word` or `kill-word` works according to the new definition.

VI. Other Enhancements

Mule is realized as patches to GNU Emacs. Enhancements are of two types: changes in GNU Emacs kernel and utilities defined by EmacsLisp. With regard to Emacs kernel changes, we have already explained about extension of character objects in EmacsLisp, user-definable character sets and coding-system and extension of character syntax. New EmacsLisp utilities include new input methods such as Tamago and Quail and better user interface to Kernel functions to enable code definition and character set definition. Here we outline other enhancements.

VI.A. Display routine

Displaying multi-lingual text requires several changes. The most important one is displaying characters itself. Mule has to change fonts (when running on X) or convert code (on terminals) whenever character sets are switched. While displaying Thai characters, Some characters are not encoded and they must be dynamically composed on demand.

In Mule, characters can have displaying attributes. Possible attributes are *inverse, underline,* and *boldface*. You have already seen the first two in the figures demonstrating inputting methods.

VI.B. Cursor movement

Cursors also requires special treatment. In GNU Emacs, every character is internally represented by one 8-bit code and occupies the same width on the screen. In Mule, however, different character hold different width and are encoded in different code length. In some languages (e.g. Hebrew), writing direction may differ.

In order to deal with these problems, character width, code length and writing direction are assigned for each leading character and Mule inspects leading characters to decide cursor movement and cursor width. A leading character also acts as an anchor to find the beginning of a character in buffer or string, as it never appears in a character code .

Table 4: Character Category

Mnemonic	Character Category
a	ASCII characters
b	Arabic characters
c	Chinese two-byte charactes
g	Greek characters
h	Hangul two-byte characters
j	Japanese two-byte characters
k	Japanese one-byte Katakana characters
l	Latin-1 characters
r	Japanese one-byte Roman characters
t	Big5 Level2 characters
w	Hebrew characters
y	Cyrillic characters

VI.C. Code conversion

Users can set and utilize different coding system while file input/output, process communication, output to terminal and input from keyboard. We provide Mule a generic ISO 2022 interpreter which converts codes according to the attributes described in the subsection II.B. Conversion to/from a coding system that does not satisfy ISO 2022 is also supported. From historical reason, Shift-JIS and BIG5 are supported by Mule kernel. For other codes, Mule has a simple stack language called CCL (Code Conversion Language) and users can define arbitrary conversion routine[8].

VI.D. Regular expression compiler

Mule provides a regular expression compiler. Regular expression compiler generates from an original regular expression two efficient regular expression forms for forward and backward search. This enables backward re-match or backward search of any text fragment represented in a regular expression. In addition, regular expression compiler can be used to define parts of text (e.g. "word").

VII. Concluding Remarks

We have developed Mule: MULtilingual Enhancement to GNU Emacs to realize multi-lingual environment for text processing. Mule handles 8-bit and 16-bit predefined or user-defined character sets and provides character input methods through ASCII keyboards. Users can define ISO 2022 variant coding systems, new input translation rules and language-specific syntax about characters and words and customize their own environments.

We are now considering the support of two new languages: Devanagari and Arabic. Future plan includes enhancement to GNU Emacs version 19 and mergence with GNU Emacs.

[8] Vietnamese VISCII is handled by CCL.

VIII. References

[1] M. Hagiya et al., "Overview of GMW+Wnn System," *Advances in Software Science and Technology*, vol. 1, pp. 133–156, 1989.

[2] R. M. Stallman, "GNU Emacs Manual," *GNU Emacs 18.59 source*, February 1988.

[3] Z. Xingguo, "A Multiple Pharase Pinyin/Hanzi Conversion Mechanism," *J. Chinese Information Processing*, vol. 4,no. 2, 1990.

Distribution and Mailing List

Mule is available by anonymous ftp from:

- etlport.etl.go.jp[192.31.197.99]: /pub/mule
  manager `<handa@etl.go.jp>`
- sh.wide.ad.jp[133.4.11.11]: /JAPAN/mule/mule-0.9.7
  manager `<kato@wide.sfc.keio.ac.jp>`
- ftp.funet.fi [128.214.6.100]: /pub/gnu/emacs/mule
  manager `<haa@cs.hut.fi>`

Access to etlport from outside of Japan is not recommended.

We are running two mailing lists:

- `mule@etl.go.jp` – main language is English
- `mule-jp@etl.go.jp` – main language is Japanese

Send requests of subscribing to:
`mule-request@etl.go.jp`
specifying which mailing list you want to join.

Acknowledgment

We thank member of mule mailing lists for their comments, ideas, bug reports and bug fixes.

Author Information

Dr.Nishikimi joined ElectroTechnical Laboratory in April 1985, where she is making a research on machine/human learning. She received her B.Eng., M.Eng., and Ph.D. from Tokyo University in 1983, 1985 and 1992.

Mr.Handa joined ElectroTechnical Laboratory in April 1983, where he is making a research on machine learning. He received his B.Eng. and M.Eng. from Tokyo University in 1981 and 1983.

Dr.Tomura joined ElectroTechnical Laboratory in April 1983. Since 1993 he has been the chief of Computer Languages Section, Computer Science Division. His research interests include programming languages and its environments. He received his B.Eng., M.Eng., and Ph.D. from Tokyo University in 1977, 1979 and 1990.