Unified and Extensible Mechanism for Multilingual Text Processing

Kenichi Handa, Mikiko Nishikimi, Satoru Tomura, Naoto Takahashi

Electrotechnical Laboratory
Umezono 1-1-4, Tsukuba, 305 JAPAN

Abstract. The most important issues for multilingual text processing systems are (1) unified mechanism for multiple languages and, (2) adjustability and extensibility of facilities provided by the system. In this paper, we explain how these issues are solved in the system called Mule (MULtilingual Enhancement to GNU Emacs). Mule is a plain-text oriented screen editor running on various systems (Unix, DOS, Windows and OS/2 on many platforms). It provides users a flexible multilingual text processing environment which covers not only simple text editing but also reading/writing Internet mail and news, browsing Web pages, etc. All these facilities are controlled by a unified mechanism and are fully adjustable and extensible by users.

1 Introduction

With the remarkable progress of computer technology, we now have sufficient computer power to handle not only English but also many other languages. Many countries have already developed ways to handle their national languages on computers. Most of them are based on locale mechanism which cause no problem as far as each computer environment is limited to its own locale. But, as the Internet connects the entire world more tightly, the need to make smooth communication among various locales/nations/languages gets larger, and we are now faced to the difficulty in handling multiple languages together. It is because each locale system has been developed independently, without concerning other locale systems.

Although there exist many projects going on such as TEI[1], EAGLES[2], and MULTEXT[3] which are trying to solve multilingual problems, most of them focus on linguistic aspects and give no solution to the more basic problem of how to represent characters of various languages. For instance, TEI provides rich format to represent a structure of text, whereas it gives no guideline how to encode each specific character.

On the other hand, non-structured plain text is currently the most widely used data for information exchange. A plain text carries only information of characters. We believe that we must, at first, create a low level mechanism to handle plain text, and then on this mechanism, make higher level applications to handle structure of text or lexical information.

Along this policy, we have developed Mule[4, 5], a multilingual plain-text editor based on GNU Emacs. GNU Emacs is one of the most famous and widely

used editor in Unix world. The biggest feature of Emacs is that it is equipped with Emacs Lisp[6] interpreter, and most of the facilities are implemented in Emacs Lisp, which means that it is possible to enhance GNU Emacs just by modifying or adding Emacs Lisp programs. Emacs can be used not only for an editing work but for almost all kinds of text processing work including reading/writing Internet mail and news, browsing Web pages, manual referencing, looking up online dictionary, etc. All these are realized by Emacs Lisp programs, and hence, are fully customizable.

Mule inherits all these facilities and can utilize them in multilingual environment. For instance, Mule users can exchange mails and read Web pages in any languages, as far as those languages are supported by Mule.

In the design of Mule, we have kept the following two points in mind:

- unified mechanism for multiple languages
- adjustability and extensibility

The most difficult but important issue of multilingual text processing is to give it a unified mechanism for handling many languages without losing the convenience realized through localization for each language. Text processing works for various languages may differ from each other in many aspects: which character set is used, how text is rendered on screen, how text is input from keyboard, what encoding mechanism is used in files. A unified mechanism which covers these varieties is inevitable to deal with multiple target languages.

Another important issues are adjustability and extensibility, in other words, how easy it is to customize or to augment facilities. It is almost impossible to design and provide all the functions necessary for all supported languages beforehand. The number of supported languages may change in the future. In developing countries, even the standard environment for national languages may change. Any system without flexibility of modifying or augmenting its facilities will soon get useless, or get too difficult to add new languages.

Therefore, even when we find that a new function is necessary for some language, we do not add it immediately. At first, we examine if there is any possibility that some other languages also need similar function, which part of the function should be generalized for all languages and which part of the function should be modifiable. Then we design the most versatile function.

Text processing is a complex work. At least it contains text inputting, storing, restoring, and displaying. When we say Mule supports some language, it means that Mule can at least:

(1) read/write files encoded in any standard format for that language,
(2) provide standard input methods for that language, and
(3) display text appropriately if we have a standard font for that language.

Here, we use the word “standard” not only for “official standard” but also for “de facto standard”. All these facilities are controlled by a unified mechanism for handling character sets, coding systems, input methods, and display routines.

In the following sections, we explain these facilities putting emphasis on how they can be customized and enhanced. Section 2 describes how Mule treats characters internally. Section 3 describes how it interacts with external world in which many different encoding mechanisms are used for multilingual text. Section 4 describes how multilingual text is input by users from normal English-based keyboard. Section 5 describes how Mule renders multilingual text on computer screen. At last, in Section 6, we summarize this paper and describe the current status of Mule and future works on Mule.

2 Internal Representation of Multilingual Text

A character set is a group of characters used together for some regional text (i.e. English text, Japanese text). Although single character set for all languages is a sweetest dream for a system developer, there is no such thing for the moment. ISO 10646[7] (or Unicode[8]) is of no use for the moment especially for Chinese characters because of its inconsistent handling of CJK-unification. That is why we decided to handle multiple character sets in Mule.

Most of Mule's character sets have one to one correspondence to character sets registered in ISO (e.g. ISO 8859-1, JIS X0208). Although each ISO's character set is identified by its *size-type* and *final-character*, Mule identifies each character set by a unique identification number (*charset-id* here after). Thus, to define a character set in Mule is to associate a unique *charset-id* to the corresponding ISO character set and to inform Mule of several parameters of the character set used for editing work such as displaying width, writing direction, etc. Table 1 shows examples of character sets.

Table 1. Example of character sets

ISO character set			parameters used in Mule			
name	*size-type*	*final-char*	*charset-id*	*bytes*	*width*	*direction*
ASCII	94	'B'	0	1	1	⇒
ISO8859-1 (Latin1)	96	'A'	129	2	1	⇒
ISO8859-8 (Hebrew)	96	'H'	136	2	1	⇐
TIS620	96	'T'	133	2	1	⇒
GB2312	94×94	'A'	145	3	1	⇒
JISX0208	94×94	'B'	146	3	1	⇒
CNS11643-1	94×94	'G'	149	4	1	⇒
CNS11643-3	94×94	'I'	246	4	1	⇒

In the table, *bytes* is byte length of Mule's internal representation, *width* is number of columns occupied on screen, *direction* is the rendering direction: left-to-right (⇒) or right-to-left (⇐).

Since all Mule's editing works (e.g. deleting a character, move a cursor forward, etc.) are done by referring to the specified parameters of each character set, they also work well for a character set added afterward.

It is also possible to define a character set which is not registered in ISO. In that case, Mule uses a *final-character* reserved for private use by ISO. If a character set does not conform to the technical requirement of ISO 2022, it should be arranged to be meet the technical requirement by dividing into small character sets, or by changing character code points. The typical example of this case is Vietnamese character set, the detail of which is described in the next section.

The most common and simplest way to hold a text in computer memory is to represent each character by array of fixed length elements (one to four bytes). Considering multilingual text, however, one or two bytes for each character is apparently insufficient to cover characters of all over the world, and four bytes element is perhaps sufficient but is a waste of memory for English only text. Since Mule is an editor, it is expected that users want to handle vary large size of text. Hence, instead of using fixed length representation, we adopted multi-byte variable length form (*multibyte form* in short here after) to represent characters in Mule's buffer both for efficient memory usage and for extensibility[1].

With *multibyte form*, each character is represented by one or two bytes of leading codes for *charset-id* and the following one or two bytes for the character code. The only exception is ASCII characters, they are represented as is, and the *charset-id* is 0. Table 2 shows more formal definition of Mule's internal character representation.

Table 2. Formal definition of the internal character representation

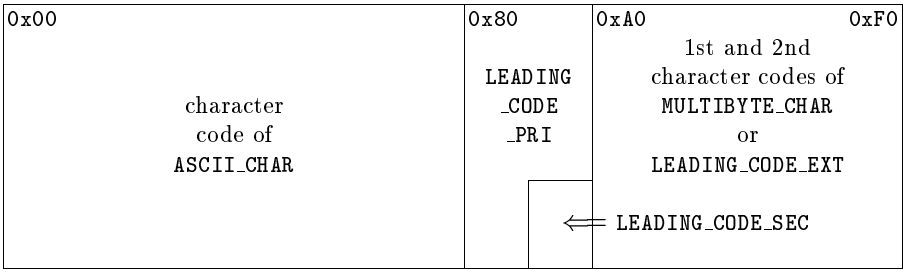
```
CHARACTER := ASCII_CHAR | MULTIBYTE_CHAR
MULTIBYTE_CHAR := PRIMARY_CHAR_1 | PRIMARY_CHAR_2
                  | SECONDARY_CHAR_1 | SECONDARY_CHAR_2
PRIMARY_CHAR_1   := LEADING_CODE_PRI C1
PRIMARY_CHAR_2   := LEADING_CODE_PRI C1 C2
SECONDARY_CHAR_1 := LEADING_CODE_SEC LEADING_CODE_EXT C1
SECONDARY_CHAR_2 := LEADING_CODE_SEC LEADING_CODE_EXT C1 C2
ASCII_CHAR := 0 | 1 | ... | 127
LEADING_CODE_PRI := 129 | 130 | ... | 153
LEADING_CODE_SEC := 154 | 155 | 156 | 157
C1, C2, LEADING_CODE_EXT := 160 | 161 | ... | 255
```

In the table, the difference between `PRIMARY` and `SECONDARY` is just a required memory per character. There is no difference in editing work between them. *Charset-id* is represented by single `LEADING_CODE_PRI` (in this case, *charset-id* is from 129 to 154) or a sequence of `LEADING_CODE_SEC` and `LEADING_CODE_EXT` (in this case, *charset-id* is more than 160). A single byte character set can contain at most 96 characters and is represented by two or three bytes sequence, a double byte character set can contain at most 9216 (96×96) characters and is represented by three or four bytes sequence. We selected most frequently used

[1] This idea is originated to Stallman's brief note[9].

character sets and defined them as PRIMARY. All character sets added by users are defined as SECONDARY. For instance, in the series of Chinese (Taiwanese) character set CNS11643, the first two plains are PRIMARY but the remaining plains are SECONDARY (See Table 1). Figure 1 shows the usage of one byte code area.

Fig. 1. Usage of one byte code area



3 Coding System

A coding system, or an encoding mechanism, is a way how to encode a text. We use many different coding systems on computer. Different countries use different coding systems. So, Mule does code conversion automatically from various representation format of text to the internal *multibyte form* described in the previous section whenever Mule interacts with outer world by reading/writing files, communicating with another process, communicating via network, accepting data from a terminal, or outputting data to a terminal[2].

To make a process of code conversion adjustable and extensible, we avoided equipping Mule with a hard-coded conversion routine. Instead, we made a generic model of coding system with several parameters to be filled in. Fortunately, most of coding systems used now in the world fit in the framework of ISO 2022[10]. Therefore, we categorized a coding system into ISO-2022 type and non-ISO-2022 type. For the former type, we created generic ISO 2022 interpreter. For the latter type, we designed a simple programming language CCL (Code Conversion Language) and created the interpreter of CCL.

When users specify some coding system for some processing (e.g. reading a file, or sending mail), Mule automatically invokes the ISO 2022 interpreter or the CCL interpreter according to the specified coding system.

3.1 ISO-2022 Type Coding System

Although ISO 2022 allows lots of variations to encode the same text, just a few of them are actually used. Thus in order to specify a encoding, small number of

[2] The conversion to/from a terminal do not occur when Mule is running on some window system

parameters listed in Table 3 are enough. For instance, Chinese, Japanese, and Korean variants of EUC (Extended Unix Code) and all ISO-8859 series (Part number 1 through 10)[11] differ only in the parameters *initial designations* and *reserved designations*. The other examples are 7-bit ISO-2022 series such as ISO-2022-JP[12], ISO-2022-JP-2[13], ISO-2022-KR[14], and ISO-2022-CN[15]. They all uses *7-bit environment*. They differ only in *reserved designations* and *locking shift*. The first two do not use *locking shift* function, whereas the remaining use the function.

Table 3. Parameters of ISO-2022 type coding system

parameter	value	meaning
initial designations	list of *charset-id*	For each graphic register, which character set is designated to initially.
reserved designations	list of *charset-id*	For each graphic register, which character set is designated to exclusively on encoding.
7-bit environment	true/false	Use only lower 7-bit or use full 8-bit on encoding.
locking shift	true/false	Use locking shift function or not.
single shift	true/false	Use single shift function or not.
direction indication	true/false	use an escape sequence of ISO-6429 to indicate writing direction on encoding.

Table 4 shows how easily we can define a coding system of ISO-2022 type in Mule. After defining a coding system, it can be specifid for any situation where code conversion is required. For instance, after the definition of the table 4, users can read/write GB files and display Chinese characters on GB terminal, and exchanging mails in GB.

Table 4. Defining Chinese EUC (so called GB) coding system in Mule

```
(make-coding-system
    '*euc-china*         ;; Name of the coding system
    2                    ;; Type, '2' means ISO-2022 type
    ?C                   ;; Mnemonic character of the coding system
    "Coding-system of Chinese EUC (Extended Unix Code)."
    t                    ;; auto-detect end-of-line type (CR, CRLF, LF)
    (list lc-ascii lc-cn ;; G0 is for ASCII, G1 is for Chinese GB2312
          nil nil        ;; G2 and G3 are never used.
          nil 'ascii-eol 'ascii-cntl nil nil nil nil))
```

3.2 Non-ISO-2022 Type Coding System

Examples of non-ISO-2022 coding system are Russian KOI-8 and Vietnamese VISCII[16]. Although KOI-8 conforms to the technical requirement of ISO 2022, the code points of characters are different from ISO 8859-5 (Latin/Cyrillic alphabet) which is the default character set for Cyrillic characters in Mule.

VISCII uses full 8-bit codes for 134 Vietnamese specific characters, which does not conform to the technical requirement of ISO 2022. Hence, in Mule, Vietnamese character set is divided into two, lowercase letters and uppercase letters, and each of them is assigned *charset-id*.

Since the generic ISO 2022 interpreter can not be used for decoding and encoding these character sets, we equipped Mule with CCL programs to do the conversion. CCL is a simple but powerful programming language suitable for writing code conversion algorithm, which means that theoretically Mule can handle any kind of coding system with an appropriate CCL program. Table 5 shows the source of CCL program to encode KOI-8.

Table 5. CCL program to encode KOI-8 coding system

```
(define-ccl-program ccl-write-koi8
 '(1
   ((read r0)
    (loop (if (r0 != 140) (write-read-repeat r0)
            ((read r0) (r0 -= 160)
             (write-read-repeat r0
     [ 32 179  32  32  32  32  32  32  32  32  32  32  32  32  32  32
      225 226 247 231 228 229 246 250 233 234 235 236 237 238 239 240
      242 243 244 245 230 232 227 254 251 253 255 249 248 252 224 241
      193 194 215 199 196 197 214 218 201 202 203 204 205 206 207 208
      210 211 212 213 198 200 195 222 219 221 223 217 216 220 192 209
       32 163  32  32  32  32  32  32  32  32  32  32  32  32  32  32])
            )))))
  "CCL program to write KOI8.")
```

4 Input Method

We have investigated existing input methods for multilingual characters and categorized them into the following four types.

(1) *key mapping*
The simplest input method which maps keys of normal English keyboards to some other characters one by one. Typical examples are Greek and Russian.

(2) *key combination*
The input method which generates a composite character by combining sequence of keys. Typical examples are European languages and Vietnamese

in which a character with a diacritical mark and a tone mark are generated from an alphabet and some symbol keys. For instance, in Vietnamese input method, a key sequence of "a^'" generates Vietnamese character 'ấ'. There also exist several Chinese input methods of this type, but they requires lots of training because of the huge number of Chinese characters.

(3) mixture of (1) and (2)
An input method which at first maps keys then generates composite character by combining the mapped keys. Typical examples are Thai and Korean. For instance, in Thai input method, all key tops are mapped to Thai consonants, vowels, or tone marks, and a key sequence of a consonant character and the following vowel or tone mark generates a composite character which puts vowel and/or tone mark on top or beneath the consonant.

(4) (3) and an external *conversion program*
This kind of method is used for inputting Chinese characters. Since there are more than ten thousand Chinese characters, it is quite difficult to remember all of the key combinations. However, just inputting phonetic characters is rather easy. So, we can create an convenient input method by letting users just input phonetic characters in (1) or (3), then turning the remaining task of generating Chinese characters on computer.
For instance, in Japanese input method, a Hiragana (Japanese phonetic alphabet) sequence is at first typed, then the sequence is converted by some *conversion program* into an appropriate mixture of Kanji (Chinese letter) and Hiragana sequence. There exist several *conversion programs*, such as Wnn, Canna, SJ3 (these are all for Japanese), and cWnn (for Chinese) which can be used from Mule. They usually use vary large dictionary and knowledge about grammar of each language to generate appropriate character sequence.

In Mule, input methods of types (1) through (3) are realized as a keyboard input translation system named Quail. Quail is given one set of translation rules (called 'Quail package') at a time and translates user input accordingly. Each translation rule consists of a key sequence and the corresponding translated text. One rule can have multiple candidates of translated text, in which case, users are prompted to select one interactively.

It is quite easy to customize a Quail package. Users can just add new translation rules or modify existing translation rules. Because of the modularity of the Quail package, adding new rules for a new language is also easy. It can be done just by making a new package with an appropriate name and defining any numbers of translation rules under the package. Table 6 shows how to make a new Quail package which simulate Caps-Lock (i.e. all lower case letters are translated to upper case letters).

Input methods of type (4) are realized as a system named Tamago. It first translates keyboard inputs into some phonetic codes (Hiragana in Japanese and PinYin or ZhuYin in Chinese) and send them to an external *conversion program* through network. It is possible to use Quail system at the first stage. However, Tamago was developed independently from Quail and has its own system named *its* to generate these phonetic codes for the moment. Since conversions can not

Table 6. Example of making Quail package "Caps-Lock"

```
;; At first, define new Quail package.
(quail-define-package "caps-lock"
                      "Caps-Lock" nil "Simulate Caps-Lock")
;; Then, define translation rules of the package.
(quail-defrule "a" "A")
  ...
```

be fully automated[3], user can select some other conversions interactively. For the moment, Tamago can use Wnn and cWnn as *conversion server*.

5 Displaying

For displaying multilingual text, we must consider two cases: running Mule from some terminal (or terminal emulator such as 'xterm', 'kterm', 'cxterm', etc.), and running Mule under a window system.

In the former case, the task of rendering multilingual text appropriately is left to terminals. Hence, what Mule does is just to send correctly encoded text to terminals. The code conversion is done according to a coding system specified for terminal output. For instance, if a user runs Mule from 'cxterm', there is no way to display any other text than English and Chinese.

When Mule is running under a window system, Mule takes responsibility of displaying multilingual text correctly. In Mule, each character set is assigned the corresponding font. A collection of mappings between all character sets vs. the corresponding fonts is named *fontset* and is the basis for displaying each character. Mule may use different fontset in different context. For instance, while reading a mail, a subject field is displayed in bold face, in which case, fontset of the collection of bold fonts are used only for the subject field.

On X window, Mule's internal character codes usually match code points of the corresponding font. For instance, Japanese character set JISX0208 can be displayed correctly by the font in which character code points follows JISX0208. But, even if this matching does not hold for a certain combination of a character set and a font, we can convert internal code points to that of a font by CCL program described in the previous section. For instance, Mule's character set for Cyrillic characters is based on ISO 8859-5, whereas a user may only has a KOI-8 font. In such a case, all he has to do is to change a mapping of a fontset so that KOI-8 font is used for Cyrillic character set, and associate an appropriate CCL program to Cyrillic character set (see Table 7).

When a user adds a support for a new language, he can simply add a mapping between the new character set for the language and an appropriate font to the existing fontsets.

[3] We need semantic analysis of contexts to determine the correct conversion, which is a hard task even for the current computer (AI) technology.

Table 7. Example of using KOI8 font for Cyrillic characters

```
;; Change font mapping in the fontset DEFAULT-FONTSET.
(set-fontset-font default-fontset lc-crl "HERE_COMES_KOI8_FONT_NAME")
(define-ccl-program ccl-x-koi8
  '(0
    ((r1 -= 160)
     (r1 = r1
   [ 32 179  32  32  32  32  32  32  32  32  32  32  32  32  32  32
    225 226 247 231 228 229 246 250 233 234 235 236 237 238 239 240
    242 243 244 245 230 232 227 254 251 253 255 249 248 252 224 241
    193 194 215 199 196 197 214 218 201 202 203 204 205 206 207 208
    210 211 212 213 198 200 195 222 219 221 223 217 216 220 192 209
     32 163  32  32  32  32  32  32  32  32  32  32  32  32  32  32]))))
  "CCL program to convert chars of lc-crl (ISO8859-5) to KOI8 font.")
;; Associate the CCL program to Cyrillic character set LC-CRL.
(x-set-ccl lc-crl ccl-x-koi8)
```

6 Conclusion

We have described the multilingual text editor Mule, stressing its adjustability and extensibility. Mule is equipped with Emacs Lisp interpreter which makes the system open. Simple Emacs Lisp programs can do all customization.

Although in this paper we have focused on the basic requirements for multilingual text, text processing is much more than character inputting. In addition to the basic functions, Mule provides various ways to faciliate text processing. The examples are character categories which classify characters as users like, and a powerful regular expression compiler which enables fast regular expression search. These functinos realize complex and customizable search commands.

Mule was released in 1993. Since then, people in the world have contributed a lot of supports for their own languages. This fact shows how widely Mule is used and how easily it can be extended. Now it can handle most of the European (including Russian and Greek) and East Asian (Chinese, Japanese, Korean) languages in addition to Thai, Vietnamese, Hebrew, Arabic, Turkish, and others. We are now working hard to support Indian languages (Devanagali scripts).

Aside from new language support, there also exist many contributed applications running on Mule, such as on-line dictionary looking-up tools, MIME encoder and decoder. The existence of these tools proves that Mule can be a multilingual workbench/environment, rather than a mere editor.

Since Mule provides lots of text processing facilities, we believe that Mule can be a basis of various systems which handle higher level information of multilingual text such as natural language processing, structured text handling as being done in TEI, and linguistic analysis of MULTEXT.

Now we are integrating Mule into the original GNU Emacs in cooperation with Free Software Foundation, the organization distributing GNU Emacs. The future release of GNU Emacs will contain Mule's multilingual facilities.

References

1. Sperberg-McQueen, C.M., Burnard, L. (Eds.): Guidelines for Electronic Text Encoding and Interchange. TEI P3:1994, TextEncoding Initiative, Chicago and Oxford (1994). <http://etext.virginia.edu/TEI.html>
2. Calzolari, N., Mc Naught, J.: EAGLES Interim Report Editors' Introduction. EAGLES Document EAG–EB–IR–2, Version of October (1994).
   <http://www.ilc.pi.cnr.it/EAGLES/edintro/edintro.html>
3. Véronis, J. (coordinator): Multext – Multilingual Text Tools and Corpor. Document MUL1. Home page. Version 0.1. (1996).
   <http://www.lpl.univ-aix.fr/projects/multext/>
4. Handa, K., *et al.*: Mule: A MULtilingual Enhancement to GNU Emacs. Proc. of INET'93, GAB-1–GAB-9, Internet Society (1993).
5. Nishikimi, M., *et al.*: Mule Home Page. Electrotechnical Laboratory.
   <http://www.etl.go.jp/Research/mulepage/
6. Lewis, B, LaLiberte, D., the GNU Manual Group: GNU Emacs Lisp Reference Manual. Free Software Foundation (1995).
   <ftp://ftp.gnu.ai.mit.edu/pub/gnu/elisp-manual-19-2.4.tar.gz>
7. JTC 1 / SC 2: Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane. ISO/IEC 10646-1 (1993).
8. The Unicode Consotium: The Unicode Standard, Versin 1.0. Addison Wesley (1991).
9. Stalllman, R.M.: Ideas for extending GNU Emacs to deal with arbitrary character sets. In file etc/CHARACTERS of GNU Emacs Version 18.59.
   <ftp://ftp.gnu.ai.mit.edu/pub/gnu/emacs-18.59.tar.gz>
10. ISO JTC 1 / SC 2: Information technology – Character code structure and extension techniques. ISO/IEC 2022 (1994).
11. ISO JTC 1 / SC 2: Information processing – 8-bit single-byte coded graphic character sets. ISO 8859-1 (1987).
12. Murai, J., Crispin, M., Poel, E. van der: Japanese Character Encoding for Internet Messages. RFC **1468** (1993). <URL ftp://ds.internic.net/rfc/rfc1468.txt>
13. Ohta, M., Handa, K.: ISO-2022-JP-2: Multilingual Extension of ISO-2022-JP. RFC **1554** (1993). <ftp://ds.internic.net/rfc/rfc1554.txt>
14. Choi, U., Chon, K., *et al.*: Korean Character Encoding for Internet Messages. RFC **1557** (1993). <ftp://ds.internic.net/rfc/rfc1557.txt>
15. Zhu, HF., Hu, DY., *et al.*: Chinese Character Encoding for Internet Messages. RFC **1922** (1996) <ftp://ds.internic.net/rfc/rfc1922.txt>
16. Vietnamese Standardization Working Group: Conventions for Encoding the Vietnamese Language. RFC **1456** (1988). <ftp://ds.internic.net/rfc/rfc1456.txt>

Appendix: Mule Distribution

Mule is distributed for free under the term of GNU GENERAL PUBLIC LICENSE. Mule Version 2.3 is available through anonymous ftp from the following sites and many other mirroring sites.

```
etlport.etl.go.jp [192.31.197.99]:/pub/mule
ftp.cs.buffalo.edu [128.205.32.9]:/pub/mule
```

This article was processed using the LaTeX macro package with LLNCS style