Multilingual vi clones: past, now and the future

Jun-ichiro itojun Hagino/KAME Project
itojun@{iijlab,kame}.net

Yoshitaka Tokugawa/WIDE Project

Outline

- Internal structures and issues in:
  - Japanized elvis
  - Multilingual nvi
- Experiences gained in asian multibyte characters support

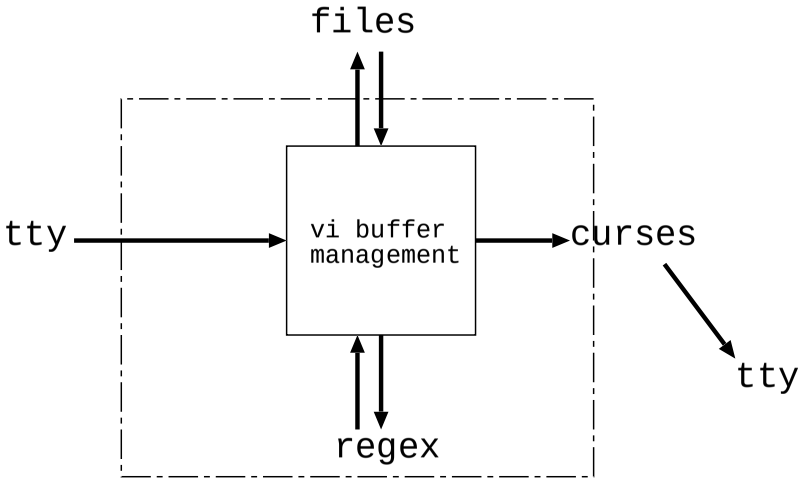
- Note: Unicode is not a solution here
  - to be discussed later

Assumptions in normal vi/vi clones

- ASCII (7bit) only, 8bit chars just go through
  - The terminal software defines interpretation
- One byte occupies 1 column on screen (except tabs)
- Assumes western languages - space between words

Architecture of normal vi

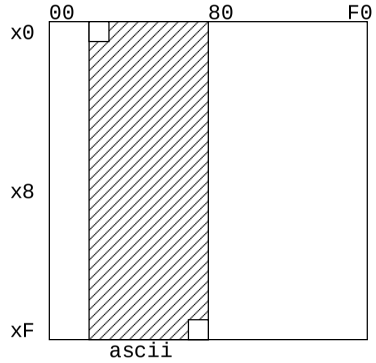
- tty input, filesystem, tty output (curses), vi internal buffer use the same encoding



Western character encodings

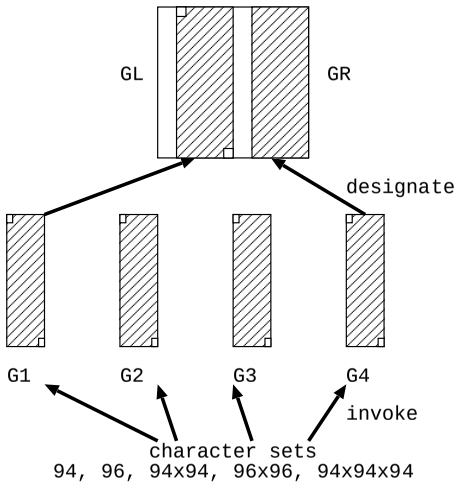
- Character encoding and the language => assumptions
  - Single byte encodings

- "ASCII" encoding
  - ASCII character set: 94 characters
- Latin 1 encoding:
  - ASCII character set
  - iso-8859-1 character set, shifted 0x80



ISO-2022 system

- Extensible character encoding system
  - By switching multiple character sets by escape sequences
  - Character set contains 94, 96, 94x94, 96x96, 94x94x94 chars
- ISO-2022 subset encodings are everywhere
  - Latin 1: fixed mapping with ASCII and iso-8859-1
  - X11 ctext



Japanese encodings

- JIS X0208 character set: 94x94 characters
- iso-2022-jp: Internet emails/netnews, UNIX
  - 41 42 43 1B 24 42 34 41 3B 7A 1B 28 42 31
- euc-jp: UNIX and other places
  - 41 42 43 B4 C1 BB FA 31
- sjis: MS-DOS and Macintosh community
  - 41 42 43 8A BF 8E 9A 31
  - Not an ISO-2022 variant

- Same character sets, different encoding method
- Single encoding is not sufficient - they all are used in various places!

Asian people needs multibyte/multilingual support

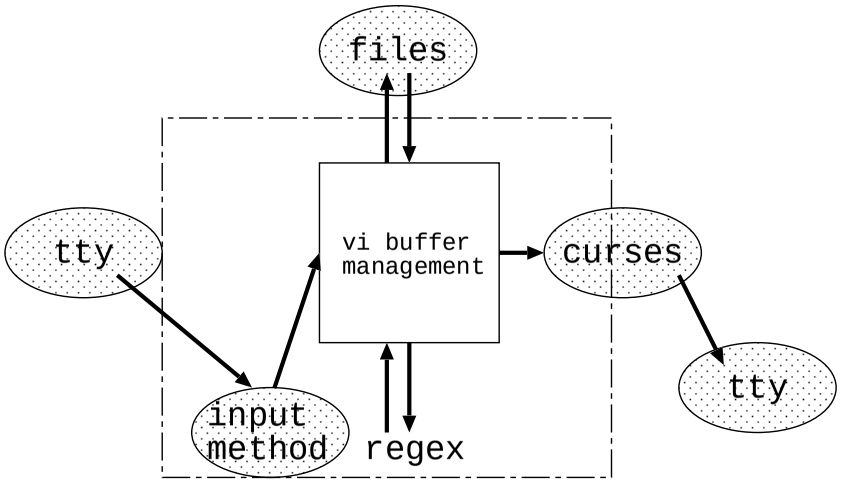
- Multibyte character sets support
  - 2 or more byte/letter
- Byte width != character width on screen
- Input methods: ondemand conversion from ASCII to multibytes
  - Use third-party libraries, like Canna or Wnn
- Switching various external encoding methods
  - For file and terminal I/O
- **Seamless** multilingual support

- => Clarify/remove the assumptions made in vi implmentation

- European people benefits from this as well
  - Handle iso-8859-x, koi8-r, and others in proper way
- Multilingual is more desirable than monolingual (Japanize)
  - Maintenance issues

architecture of multilingual vi

- Can’t assume single encoding
- Need input method (inside or outside vi)
- Must be able to switch encodings
  - tty input, input method, filesystem, tty output can use different encoding
- Internal encoding is the key issue



Design goals: What is "seamless"?

- No "Chinese mode" nor "Japanese mode" in the editing session
- Any character set can be mixed in a text, without twist
  - Some of character encodings can accomodate, say Chinese, Korean and Japanese character sets at the same time
  - Mixed language texts - Chinese document annotated with Japanese
- Preserves information in the file
  - No implicit conversion/translation
  - Implicit conversion confuses user, and it does not match the vi design
  - If you need conversion, use `:!`
- Behaves just like normal vi, over multilingual characters
  - regex, cursor movement, whatever

"jelvis" - Japanized elvis

- First generation of implementation
- Based on elvis by Steve Kirkendall

- Internal encoding: euc-jp
- External encoding: iso-2022-jp, euc-jp, sjis

- Internal encoding: 41 42 43 B4 C1 BB FA 31

- Internal encoding bytewidth == screen width
  - 2 bytes, 2 columns

- Maintenance/synchronization problem with kelvis/celvis
  - => Multilingual implementation is desirable

"nvi-m17n" - multilingualized nvi

- Current generation of implementation
- Based on nvi by Keith Bostic

- Internal encoding: internal multibyte encoding
  - ASCII is 1 byte
  - 0x80-0xff are "multibyte tag" character
  - This is similar to Mule (multilingual emacs)
- External encoding: any of iso-2022 variants, and others

- Internal encoding: 41 42 43 88 34 41 88 3B 7A 31
  - "88" is the tag for JIS X0208 Kanji character set

- Internal encoding bytewidth != screen width

Additional features

- Switching I/O encoding:
  - `:set fileencoding=iso-2022-jp`
  - `:set inputencoding=big5`
  - `:set displayencoding=euc-tw`

- Input method support: "Canna" library from NEC
  - `:set cannaserver=server.itojun.org`
  - `:set cannakey=^O`

Word boundary issues

- Asian words are not separated by spaces!

- Define word movement over Asian characters
  - The exact "word" movement requires syntactic analysis and dictionary lookup (very hard)

- Define character classes
  - Kanji letters, hiragana letters, western, symbols
- Define movement over word boundary
- Solves problem for most of the cases

萩野はFreenix会場にいます!!

- Need for explicit language information

Regex library

- Some of regex library uses 2^7 as flag bit
  - Separate flag bit from the characters
- Character range ([a-z0-9]) as bitmap
  - Impossible for multibyte chars/multilingual internal code
  - Bitmap for ASCII, start-end for others
- Metacharacter (.) must match against single multibyte char

Curses library

- Store character set information into screen buffer
- Render accordingly on redraw
  - Chararcter set
  - Character data (multibyte)
  - Offset from the beginning of the glyph
- Multi-width characters support
  - Need to erase right half, when left half is overwritten

- Multibyte with **`addch()`** is cumbersome, use **`addstr()`**
  - Intermediate state is hard to manage

Unicode as internal encoding?

- Unicode characteristics:
  - Well documented external multibyte encoding (UTF8/16)
  - 16 or 32bit fixed wide char for internal encoding (UCS2/4)
- Asian characters are "unified"
  - Some of Chinese/Korean/Japanse characters are mapped into single Unicode codepoint
  - As different characters are mapped into single codepoint, information will be lost (inverse conversion is impossible)
  - Language tagging -> "fixed-width wide char" is impossible
- Unicode is useful for "monolingual" asian processsing
  - For example, ASCII + Chinese only
  - Or, modal support like "Chinese mode" or "Korean mode"
- Unicode is not useful for multilingual processing
- Additional Unicode support would be good
  - Unicode as a character set we support, not as the internal encoding

nvi-m17n: next generation

- Use wide char (wchar_t) for internal code
  - ISO/JIS standards suggest wide char
  - Memory is now cheap
- Can’t really rely upon vendor’s locale library
  - Too little support for stateful multibyte encodings
- Need massive modification to various places
  - Support for multiple encoding in locale library
  - Support for wide char in curses/regex/whatever
- Feedback modified locale library to the community
- Add Unicode support
  - Supply file converter as external tool

Wide character library: status

- Wide char library is not really ready
  - curses, regex
  - Need support for column width query (for curses)
- Bugs in vendor-supplied locale library
  - Not heavily tested?
- Changing from char to wchar_t is a big leap for the source code tree

- glibc
  - Assumes Unicode (no support for stateful encodings), single encoding in a program
- runelocale library
  - Encoding switchable by **`$LANG`**, no support for stateful encodings, single encoding in a program

Observation

- Normal vi
  - 1byte/char
  - Single encoding (= ASCII)
- Japanized vi (jelvis)
  - Multibyte/char, bytewidth == width on screen
  - Multiple encoding in a program
- Multilingual vi (nvi-m17n)
  - Multibyte/char, bytewidth != width on screen
  - Multiple encoding in a program
- Next multilingual vi
  - Wide char, bytewidth != width on screen
  - multiple encoding in a program

- Multilingualization = less assumptions!

Future work

- Provide modified runelocale library separately to *BSD
- Right-to-left languages
- Support for other input method: cWnn (Chinese Wnn)

References

- mailing list: **nvi-m17n@foretune.co.jp**
  - discussions are (at this moment) mainly in Japanese language, questions in English are welcome
- **ftp://ftp.foretune.co.jp/pub/tools/jelvis/**
- **ftp://ftp.foretune.co.jp/pub/tools/nvi-m17n/**
- Ken Lunde, "CJKV information processing", O'reilly