An Interactive, Text-based Translation Aid Environment

Eric Nichols and **Yuji Matsumoto**
Graduate School of Information Science,
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara, 630-0192 JAPAN
{eric-n,matsu}@is.naist.jp

1 Introduction

Translation is an essential and important part of human communication. However, it is a very challenging task, requiring the translator to have a complete understanding of and a deep familiarity with the source and target languages. This difficulty is not eased by the fact that fundamentally it is an inherently repetitive task, consisting of looking up unfamiliar words and doing large amounts of editing to produce a good translation.

Given these demands, computers provide a good way to ease this repetitiveness by automating lookup and editing; by converting resources like dictionaries, other translations, and collections of example sentences to a computer-readable format, lookups can be performed much faster. Likewise sophisticated editing languages can reduce the time and complexity of editing translations.

Since the idea of a translation aid environment was first proposed by Martin Kay (1980), a number of different systems have been built. There have been systems built on (Wordfast[1]) or mimicking (Trados[2]) Word processors; stand alone applications like Déjà Vu X[3] and OmegaT[4]) and web-based translation aid systems, both free (TRANS-Bey (Bey et al., 2006)) and commercial (Lingotek[5]).

A survey of professional translators in 2006 showed that 82.5% of translators own and use at least one translation aid environment (Lagoudaki, 2006). Yet existing translation aid software has a reputation for being hard to use. The conductor of this survey summarized her findings saying users want ”flexibility, simplicity, and ease of use” in their translation aid environments. Why do current translation aid systems lack these qualities?

We consider this to be in interface problem: most translation aid environments are modeled after word processors, with functionality for processing text often hidden away in complex hierarchical menus or behind obscure keyboard shortcuts, making it difficult for translators to explore and manipulate language. Given that translation is an inherently text-centered task, it makes sense to investigate the potential of text as the interface. The Acme text editor provides an ideal environment for this goal.

2 Text as the Interface

2.1 Acme: An Interactive Text Editor

Acme is ”a hybrid of window system, shell, and editor” created by Rob Pike (1994) as a programming environment. At first glance, Acme seems like a strange editor; it is designed for use with a three-button mouse, and there are no menus or buttons of any kind. However, the interplay between the mouse and text in Acme give it a powerful and flexible interface.

2.1.1 The Mouse

In Acme, the mouse allows the user specify how to contextually interpret text. Acme is designed for use with a three-button mouse, and each button has a different use. The left button selects text. The middle button interprets it as a command. For example, clicking on the text ’`New`’ opens a new buffer window. Clicking on ’`Del`’ closes that buffer. All common editor functionality is handled in this way: ’`Snarf`’ is Acme’s copy and ’`Put`’ saves the contents of the current buffer; ’`Undo`’ and ’`Redo`’ provide their namesakes’ editing functions. Text for these commands is provided in the ”tag” area at the top of each buffer, but this is simply for convenience; text can be added and executed anywhere in Acme.

When the text middle-clicked on is not a built-in command of the Acme editor, it is interpreted as a system command, and the command is piped to a system call for execution with results are sent back to Acme. What this means is any command that operates on standard I/O can be executed just by middle-clicking on its name. In this way, Acme can be extended at any time by a large number of commands that the user is already familiar with. Outside of the small set of Acme-specific editor commands, there is no need to learn any special commands to add new functionality to Acme; almost all functionality is outsourced.

The right-button provides another useful function: it performs a context-sensitive ”get” function. When the name of an existing file is right-clicked on, it is opened in a new buffer; when the name of a function is clicked on, its definition is shown. Clicking on a number followed by a colon (’`:`’) causes the cursor to jump to that line number

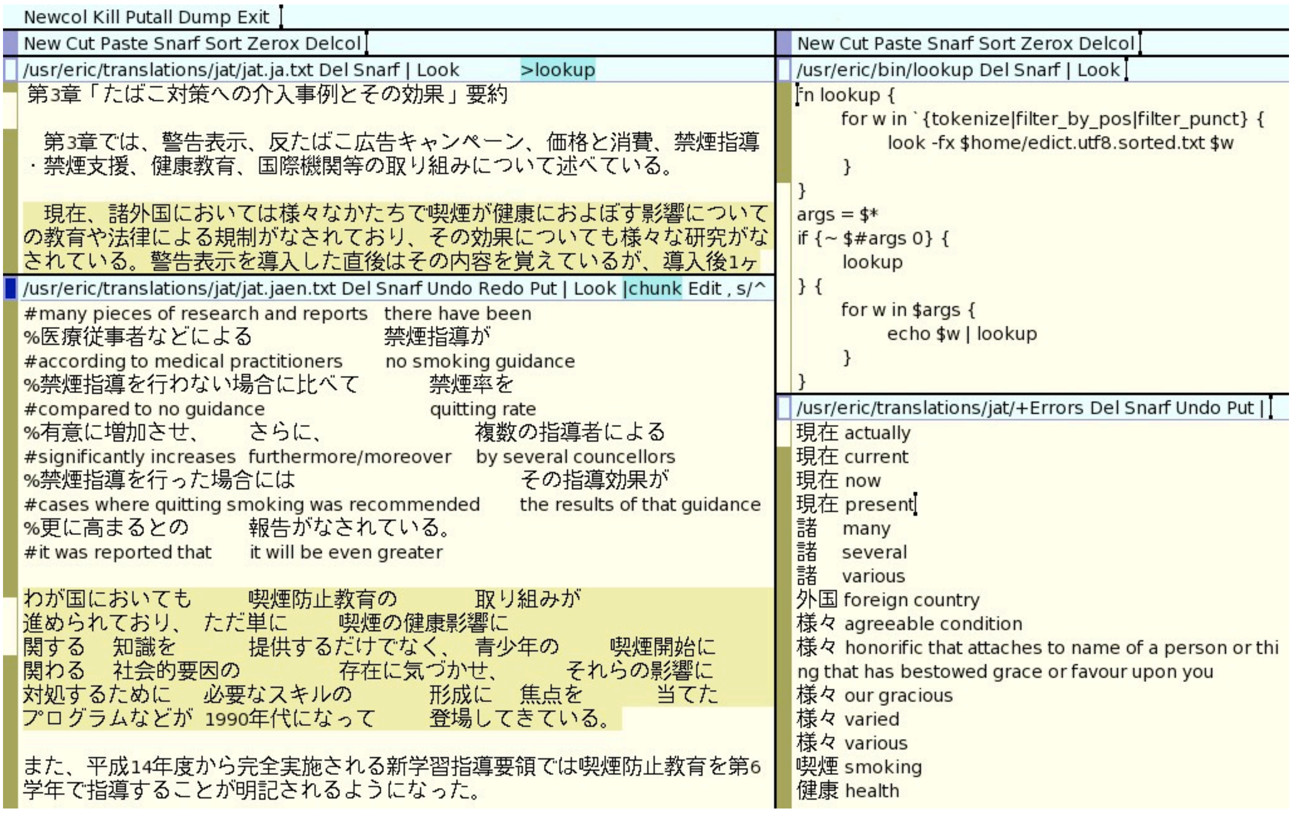
[1] Wordfast: `http://www.wordfast.net`

[2] SDL Trados: `http://www.trados.com/en/`

[3] Déjà Vu X: `http://www.atril.com/default.asp`

[4] OmegaT: `http://www.omegat.org/`

[5] Lingotek: `http://lingotek.com/features.html`


Newcol Kill Putall Dump Exit
New Cut Paste Snarf Sort Zerox Delcol
/usr/eric/translations/jat/jat.ja.txt Del Snarf | Look >lookup
/usr/eric/translations/jat/jat.jaen.txt Del Snarf Undo Redo Put | Look |chunk Edit , s/^
New Cut Paste Snarf Sort Zerox Delcol
/usr/eric/bin/lookup Del Snarf | Look
/usr/eric/translations/jat/+Errors Del Snarf Undo Put |

第3章「たばこ対策への介入事例とその効果」要約

　第3章では、警告表示、反たばこ広告キャンペーン、価格と消費、禁煙指導・禁煙支援、健康教育、国際機関等の取り組みについて述べている。

　現在、諸外国においては様々なかたちで喫煙が健康におよぼす影響についての教育や法律による規制がなされており、その効果についても様々な研究がなされている。警告表示を導入した直後はその内容を覚えているが、導入後1ヶ

```
#many pieces of research and reports   there have been
%医療従事者などによる                  禁煙指導が
#according to medical practitioners    no smoking guidance
%禁煙指導を行わない場合に比べて            禁煙率を
#compared to no guidance                   quitting rate
%有意に増加させ、    さらに、              複数の指導者による
#significantly increases  furthermore/moreover   by several councellors
%禁煙指導を行った場合には                      その指導効果が
#cases where quitting smoking was recommended     the results of that guidance
%更に高まるとの      報告がなされている。
#it was reported that     it will be even greater
```

```
わが国においても        喫煙防止教育の          取り組みが
進められており、 ただ単に       喫煙の健康影響に
関する  知識を          提供するだけでなく、  青少年の      喫煙開始に
関わる  社会的要因の                存在に気づかせ、        それらの影響に
対処するために   必要なスキルの            形成に   焦点を          当てた
プログラムなどが 1990年代になって          登場してきている。
```

また、平成14年度から完全実施される新学習指導要領では喫煙防止教育を第6学年で指導することが明記されるようになった。

```
fn lookup {
        for w in `{tokenize|filter_by_pos|filter_punct} {
                look -fx $home/edict.utf8.sorted.txt $w
        }
}
args = $*
if {~ $#args 0} {
        lookup
} {
        for w in $args {
                echo $w | lookup
        }
}
```

```
現在 actually
現在 current
現在 now
現在 present
諸   many
諸   several
諸   various
外国 foreign country
様々 agreeable condition
様々 honorific that attaches to name of a person or thi
ng that has bestowed grace or favour upon you
様々 our gracious
様々 varied
様々 various
喫煙 smoking
健康 health
```

Figure 1: Acme being used to translate a Japanese document into English

in the current buffer. Finally, if no interpretation of the text can be found, Acme searches the current buffer for another occurrence of that word.

2.1.2 The Buffer

Acme is arranged into resizable columns of buffers. Acme's buffers consist of a light blue strip of text at the top called a ”tag,” a large text are immediately below, and a scroll bar to the left of the text area. The tag area contains the current file or directory that is open. When a directory is open in Acme, the names of its contents are displayed. To the right, the text for common editing operations is displayed followed by a blank area meant for use as scratch.

Where the tab and scrollbar intersect, there is a layout box. This box acts as an anchor allowing the user to drag and reposition the buffer by left-clicking on it. Right clicking on the box minimizes the buffer, shrinking it to just its tag. Middle clicking on the box maximizes the buffer by hiding all other buffers in the same column. Clicking on the layout box makes the other buffers visible again. When a buffer is modified, its layout box turns dark blue to indicate there is unsaved data, and the 'Put' command helpfully appears in the tag.

3 Building a Translation Aid Environment with Acme

We are currently working on a small, proof-of-concept translation aid system using Acme focusing on the Japanese-English language pair. We are basing our system on Acme SAC[6], a platform-independent Acme port by Caerwyn Jones. Acme SAC provides a convenient way to distribute Acme to users bundled as a simple, click-and-run application.

Acme SAC is built on a virtualized operating system called Inferno (Pike et al., 1997). Inferno has several features that make it an ideal platform. Inferno is a hosted operating system that can run on top of Windows, Mac OS X, and Linux without any modification. The operating system is built from the ground up supporting Unicode, and it provides a full suite of Unix-style command line utilities that can be applied to text of any language. Inferno interfaces with the host OS, allowing tasks like multi-lingual input to be handled by the host's IMEs[7]. Finally, Inferno can access the data and commands on its host operating system and share those with other instances of Inferno, providing a transparent way of sharing networked resources and conducting parallel processing.

[6] Acme Stand Alone Complex (http://code.google.com/p/acme-sac)

[7] Multi-lingual input is not yet fully supported under Linux.

```
fn setlang {
    args = $*
    or {~ $#args 1 2} {
        echo 'usage: setlang <command> [-e | -j]' >[1=2]
        raise usage
    }
    (cmd args) = $*
    # set $lang if first argument is recognized language flag
    (and {~ $#args 1} {~ $args -e -j} {
        (lang nil) = $args
    })
    # pipe portion of text to whichlang for identification
    # and set $lang if it is still unset
    tmp = ${pid}^.tmp
    if {~ $#lang 0} {
        l = `{tee $tmp | sed 200q | whichlang}
        if {~ $l en} {
            lang = -e
        } {~ $l ja} {
            lang = -j
        }
    }
    # call command; pipe $tmp to command if $tmp file exists
    if {ftest -f $tmp} {
        ($cmd $lang) < $tmp
        rm -f $tmp
    } {
        $cmd $lang
    }
}
```

```
fn tok_en {
    geniatagger >[2=] |      # tokenize discarding stderr
    cut -f1 |
    tr '\n' ' '
}

fn tok_ja {
    tcs -f utf-8 -t euc-jp |
    mecab -Owakati |         # suppress POS output,
                             # tokenizing into words
    tcs -f euc-jp -t utf-8
}

fn tok_any {
    args = $*
    if {~ $1 -e} {           # English
        f = tok_en
        (lang args) = $args
    } {~ $1 -j} {            # Japanese
        f = tok_ja
        (lang args) = $args
    } {
        f = tok_en           # Use English as fallback
    }
    $f $args                 # call function on args
}

fn tokenize {
    setlang tok_any $*
}
```

Figure 2: An interface for the `tokenize` task, implemented in Inferno's shell

4 Task-driven Interfaces for Translation

4.1 Splitting Text into Translation Units

Splitting text into easy-to-handle chunks called ”Translation Units (TUs)” is a common task in a translation workflow. It often acts as a first step for alignment, translation auto-complete, and other tasks.

We implemented a command, `chunk`, which splits unprocessed text into phrase-like TUs. We use tabs as the delimiter, turning the process of editing, splitting, or merging TUs from a potentially complex operation to a simple text edit. `chunk` is implemented using the dependency parser Cabocha (Kudo and Matsumoto, 2002) for Japanese and the part-of-speech tagger and chunker Genia Tagger (Tsuruoka and Tsujii, 2005) for English.

An example application of `chunk` is shown in Figure 1: the `|chunk` command highlighted in the tag bar of the lower left-hand buffer has been called on the selected paragraph in the middle of the buffer. The ”|” symbol indicates that the `chunk` command should pipe its I/O to and from the selected region, performing an inline replace of the original text with its equivalent TUs.

4.2 Dictionary Lookup

Consultation of references is an essential function of any translation aid environment. Acme provides a dictionary browsing mode called '`adict`' that displays a word's full definition and allows easy navigation to other entries. We also provide a batch lookup command that lets the user retrieve the definitions of all of the words in a section of text. To look up words the input is tokenized and lemmatized, and all of the resulting tokens are searched for in the dictionary. The definitions are displayed in a new buffer window in the order of the input, providing some context for choosing a translation. We currently use Edict[8] and Wikipedia[9] for our dictionaries.

Batch lookup is shown in action in Figure 1: the command `>lookup` in the upper left-hand buffer has been applied to the selected paragraph of text. The results of the lookup appear in the lower rightmost buffer. The ”>” symbol indicates that the selected text is piped to the command as input.

4.3 Editing and Formatting

Acme provides a powerful, regular expression based editing language called `Edit`. `Edit`'s command language resembles that of the Unix command `sed`. It has been extended to allow the user to specify target regions of text either manually or through pattern matching. A typical call may look like this: `Edit , s/^%//g`.

The comma instructs `Edit` to apply the command to the entire buffer; the remaining command matches lines starting with a percent sign and removes it, in effect uncommenting them. In this way, `Edit` obviates the need for a ”search-and-replace” function, like commonly built into other text editors.

Edit can be used to apply any command to the text in a buffer. For example `grep -v` could be used to delete lines matching a certain pattern, or a programming in a scripting language could be used for more intricate processing.

[8] Edict: `http://www.csse.monash.edu.au/~jwb/j_edict.html`

[9] Wikipedia: `http://www.wikipedia.org/`

5 Consistent Interfaces for Disparate Tools

Our goal is to provide translators with a simple, consistent, language-independent interface for each task regardless of the complexity of underlying implementation. We do this by abstracting away language-dependent implementations and selecting the proper one based on the input language.

We start by standardizing the input and output for each task. For example, `tokenizer`, as shown in Figure 2, takes unmodified text as its input and returns text tokenized at the word level and delimited by whitespace.

Next, each language-specific implementation is given its own shell function named *task_language*. `tok_en` shows how the English tokenizer is implemented by filtering and reassembling the output of `geniatagger`.[10] The corresponding Japanese function, `tok_ja` is implemented by using the `-Owakatigaki` output mode of the part-of-speech tagger MeCab[11]. `tcs` is a command line utility that handles the conversion to and from the EUC-JP encoding necessary to use `mecab`.

All of the language-specific implementations are then gathered in a shell function named *task_all* that acts as a multiplexer, selecting the proper implementation by a language flag that is set when the function is called. `tok_all` defaults to English if a supported language is not specified.

Finally, the complete user interface is produced by wrapping the multiplexer with the `setlang` function. `setlang` checks to see if a language flag has been manually set, and if not, pipes a small portion of the input to `whichlang`, a simple language auto-identification heuristic whose implementation is described in (Nichols and Matsumoto, 2007).

The final user interface is a simple one-word command: `tokenize`. The user simply highlights the text to be processed and middle-clicks on the text `tokenize` to execute it. This method of abstraction is powerful and reusable; we use it to build the interfaces of `chunk`, `lookup`, and the other translation tasks.

6 Conclusion

6.1 The Power of Text

Although it is still early in its developmental phase, Acme shows promise as translation aid environment, and we are already using it for translating software manuals in our lab. By embracing text as its interface, Acme removes the barrier between users and their text, making it easier to apply their existing tools and resources, and giving them the freedom needed to adapt to dynamic workflows. As we saw in Section 4, adopting a Unix pipe-based model of text processing simplifies both the interfaces of NLP tools and their integration into the translation aid environment.

6.2 Problems Encountered

There are several problems that must be addressed before Acme can become a fully-usable translation aid environment. (1) One of Acme's strengths is the ease in which existing tools can be incorporated, however, tools external to Acme and Inferno still need to be installed by the end user. This can be difficult for programs like MeCab and Genia Tagger, as the installation requirements and process can vary between host OSes. (2) Furthermore, it is argued in (Abekawa and Kageura, 2007) that vocabulary lookup needs to be tailored to the task of translation to be effective. We need to consider whether off-the-shelf NLP components are really up to this task. (3) Finally, as most translators cannot avoid formatted documents like MS Word and PDFs, we need to explore how markup can best be handled in a plain-text based environment.

6.3 Future Work

We have several ideas for improving Acme's usefulness as a translation environment. (1) In order to solve software distribution problems and create tools that are more appropriate for translation, we plan to implement a language-independent tokenizer and chunker that will run natively in Inferno. This will allow us to package all of the translation aid software as a part of Acme SAC. (2) Because Acme only handles Unicode text, we currently must deal with constantly re-encoding the text as different tools are applied. Noah Evans (2007) has developed a file system layer that will transparently handle conversion to and from Unicode when files are accessed in Acme. (3) We plan to explore how Inferno's network sharing capabilities could be used to allow translators to pool their resources.

References

Takeshi Abekawa and Kyo Kageura. 2007. A translation aid system with a stratified lookup interface. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Demos and Posters*, pages 5–8, Morristown, NJ, USA. Association for Computational Linguistics.

Youcef Bey, Christian Boitet, and Kyo Kageura. 2006. The transbey prototype: An online collaborative wiki-based cat environment for volunteer translators. In *Proceedings of the 3rd International Workshop on Language Resources for Translation Work, Research & Training*, pages 49–54. Fifth International Conference on Language Resources and Evaluation.

Noah Evans. 2007. Representing disparate resources by layering namespaces. In *Proceedings of the Second International Workshop on Plan 9*, pages 19–24.

Martin Kay. 1980. The proper place of men and machines in language translation. *Technical Report CSL*, 80-11.

Tako Kudo and Yuji Matsumoto. 2002. Japanese dependency analyisis using cascaded chunking. In *Proceedings of CoNLL-2002*, pages 63–69.

Elina Lagoudaki. 2006. Translation memories survey 2006: Users' perceptions around tm use. In *Proceedings of the ASLIB International Conference Translating & the Computer 28*.

Eric Nichols and Yuji Matsumoto. 2007. Acme as an interactive translation environment. In *Proceedings of the Second International Workshop on Plan 9*, pages 35–46.

Rob Pike, Dave Presotto, Sean Dorward, Dennis M. Ritchie, Howard Trickey, and Phil Winterbottom. 1997. The Inferno operating system. *Bell Labs Technical Journal*, 2(1), Winter.

Rob Pike. 1994. Acme: A user interface for programmers. In *Winter 1994 USENIX Conference*, pages 223–234.

Yoshimasa Tsuruoka and Jun'ichi Tsujii. 2005. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 467–474, Morristown, NJ, USA. Association for Computational Linguistics.

[10] `>[2=]` is a shell idiom to disable standard error output.

[11] MeCab: `http://mecab.sourceforge.net/`