SuprIME: IME によるテキスト編集機能の統合

中園 翔　　増井 俊之 *

概要. 計算機上で様々なテキストエディタが利用されているが，システムごとに編集方法や編集機能が異なっているのが不便である．本論文では，各国語入力のために OS に用意されている IME(Input Method Editor) 機能を利用することにより，あらゆるテキストエディタにおいて同じ操作によるテキスト編集を可能にする方法を提案する．我々の手法を利用すると，異なるテキストエディタ上での編集操作が共通化されるだけでなく，テキスト編集時に便利な様々な機能をあらゆるエディタで利用することが可能になる．

1 はじめに

計算機上でテキストを編集するために様々なテキストエディタが利用されている．文書を作成するときはワープロを利用し，メールを書くにはメールクライアントを利用し，文字端末でのプログラム開発には vim や Emacs を利用し，IDE を利用した開発では付属のエディタを利用し，ネット上でテキストを扱うにはブラウザのテキストフォームを利用するといったように，場合に応じて様々なエディタが利用されている．

エディタの機能や操作体系はエディタごとに異なっているのが普通である．ブラウザやワープロでテキストを 1 行消したい場合はマウスで行全体を選択してから削除キーを押せばよいが，vim では「d」キーを 2 回タイプして消すのが普通であり，Emacs では Ctrl-K キーが利用される．Emacs に慣れたユーザがワープロ上でも Ctrl-K で行を消去したいと思っても，そのようなカスタマイズはできないのが普通であるし，機能拡張が可能なシステムを利用している場合でも，操作体系を完全に同じにすることは難しい．

あらゆるエディタの操作を統一することは難しいが，様々なエディタで共通に利用できるソフトウェア層をユーザとアプリケーションの間に置くことができれば，異なるエディタの編集操作をある程度共通化できる可能性がある．現在のパソコンには IME(Input Method Editor) と呼ばれる文字入力機構が用意されており，様々な言語のテキスト入力に利用されている．IME はエディタなどとは独立したソフトウェアであり，ユーザのすべてのキー入力を受け取って各国語に変換した結果をアプリケーションに送出する．IME はあらゆるアプリケーションで共通に利用されるので，たとえば日本語入力用の IME を利用する場合，Emacs でもブラウザでも IDE でも同じ操作で日本語を入力できる．IME は一般には各国語入力のみのために利用されているが，テキストの挿入/移動/削除といった編集操作も IME が受け持つようにすれば，様々なエディタ上で同じ操作で編集を行なうことが可能になると考えられる．

このような考えにもとづき，Mac 上の様々なテキストエディタにおいて同じキー操作によるテキスト編集を可能にする SuprIME システムを試作した．本論文では SuprIME の実装と利用方法について述べ，柔軟でユニバーサルなテキストエディタの構築について考察する．

2 SuprIME 使用例

2.1 日本語入力

SuprIME は，MacRuby[1] で記述された日本語 IME である「Gyaim[2]」に様々な編集機能を追加したものである．図 1 は SuprIME による日本語入力の例である．IME はアルファベット以外の文字を入力するときだけ有効にするのが普通であるが，SuprIME は，編集操作を行うために常に有効であることを想定して実装した．そのため、英語入力と日本語入力の両方が可能である．

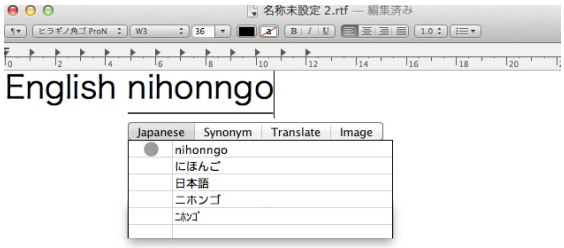

図 1. SuprIME による日本語入力の例

* 慶應義塾大学環境情報学部 環境情報学科，慶應義塾大学環境情報学部

[1] Ruby を Mac 用に拡張したもので，あらゆる Mac の API を Ruby から利用することができる．

[2] https://github.com/masui/Gyaim

2.2 ブロック移動

テキストの一部を別の場所に移動する操作はテキストエディタの重要な機能のひとつであるが，エディタによって操作方法が大きく異なっている．たとえばEmacsでテキストを移動させたい場合は，移動する領域をキー操作によって指定してから削除/コピーし，カーソルを移動してからペーストするという手順を利用するのに対し，ブラウザの編集領域でテキストを移動させたい場合は，マウスで領域を指定した後で選択領域をドラッグして別の位置に移動することが多い．このように，テキスト移動のような基本操作でもエディタごとに操作が異なっているのは不便であり，操作ミスをしがちであるが，SuprIMEを利用するとあらゆるエディタにおいて同じ操作でテキストを移動することができる．

図2はMac OSに標準登載されている「テキストエディット」でテキストを編集しているところである．

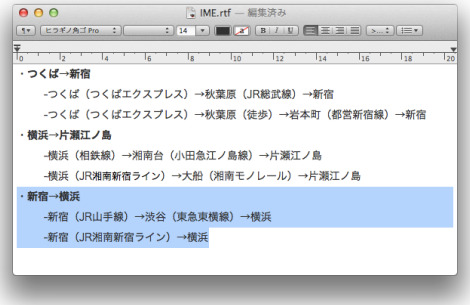

図 2. ブロック移動前の状態

ここでShift+↑キーを押すとテキストは図3のように変化する．

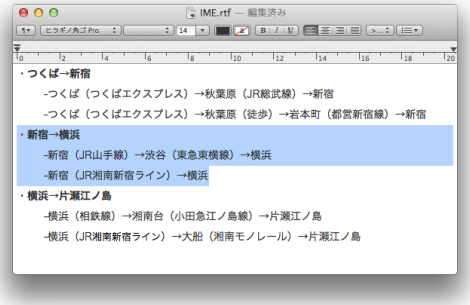

図 3. Shift+↑キーを押した後の状態

図4はブラウザ上でGoogle Docsのテキストを編集しているところである．ここでShift+↓キーを二度押すと，テキストは図5のように変化する．

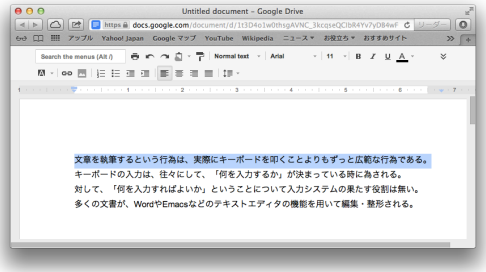

図 4. ブロック移動前の状態

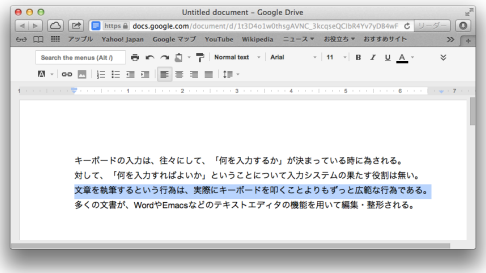

図 5. Shift+↑キーを押した後の状態

このように，あらゆるエディタにおいてShift+矢印キーという共通の操作でブロック移動を行なうことができる．

2.3 連続インデント

前述したブロック移動は，インデントを調整する機能も備えている．プログラミングを行なっている時だけでなく，普通のテキストを編集している場合でも行頭の空白やタブの量を調整するインデント処理は頻繁に行われるが，自動的にインデントを行うエディタもあれば，コマンドを打ち込まなければならないものもあり，そのような調整機能を持っていないエディタも多い．SuprIMEを利用すると，あらゆるエディタや入力フィールドでブロック移動と同時にインデントが自動で行われる．

図6は，Xcode上でPythonプログラムを書いている例である．Xcodeは，Objective-CやRubyなどの言語で自動インデントの機能を備えているが，Pythonには対応していない．たとえば，図6のようなテキストに対して図7のように移動させたいブロックを選択し，Shift+↓キーを入力することで，テキストは図8のように変化する．

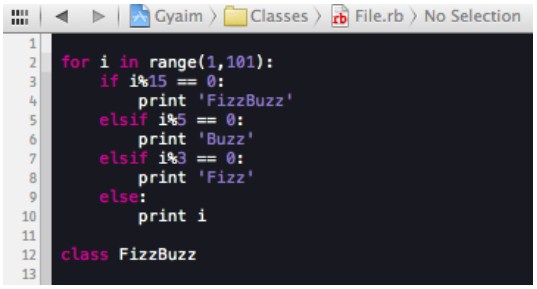

図 6. ブロック移動前の状態

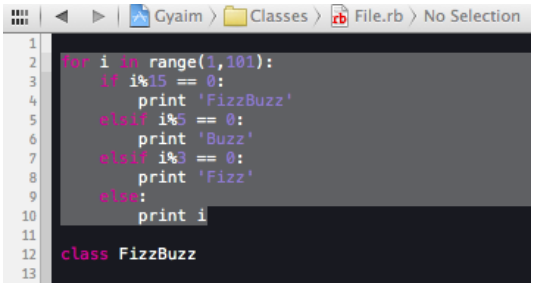

図 7. テキストを選択した状態

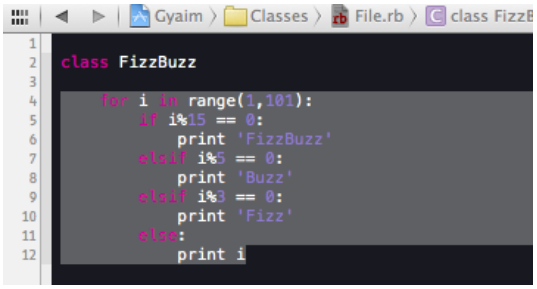

図 8. ブロック移動後の状態

このように，あらゆるエディタにおいて，ブロック移動を行う際に適切な空白とタブを自動で補完することができる．

2.4 単語置換

多くのエディタが単語の検索/置換機能を持っているが，その操作方法はシステムごとに異なっている．例えば，コンソール上で検索と置換を行う時は grep や sed コマンドを用いるが，多くのエディタソフトでは Control+F や Command+F が用意されている．

SuprIME を利用することでこの操作を共通化する例を挙げる．

あらかじめ割り当てたファンクションキーを押すと，図 9 のように SuprIME の単語置換ウィンドウが起動する．

図 9. 単語置換ウィンドウの例

図 10 のように，入力語と置換語をテキストエリアに打ち込み，Replace ボタンを押すと，

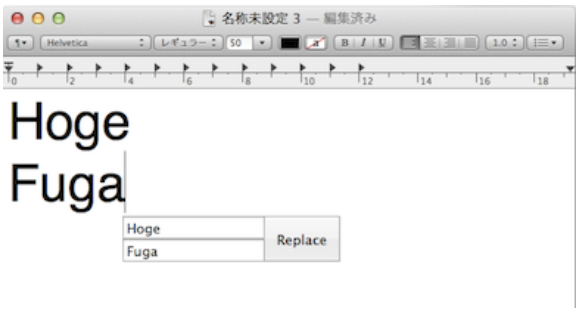

図 10. 置換ウィンドウに単語を入力した例

テキストは図 11 のように置換される．

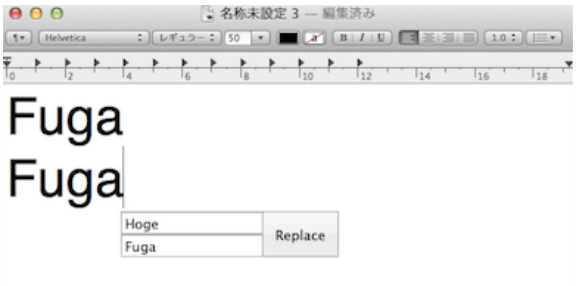

図 11. 単語置換後の状態

このように，あらゆるエディタ上で検索と置換が可能である．

2.5 Dynamic Macro

Emacs では lisp プログラムでエディタの機能を拡張することが出来るが，これらのスクリプトを IME 上に実装することで，他のエディタ上でも Emacs と同様の拡張機能を動作させることができる．

以下に SuprIME で Dynamic Macro[7] を実現した例を挙げる．Dynamic Macro は，入力の繰り返しを自動化する Emacs 拡張であるが，SuprIME ではこれと同様の機能を Ruby で実装している．図 12 のように，ユーザの入力操作が繰り返しになってい

るとき，任意に設定したファンクションキーを押すと，テキストは図 13 のように編集される．

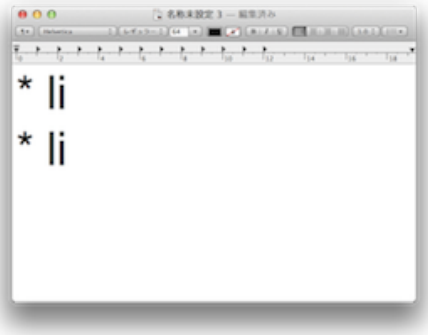

図 12. Dynamic Macro 機能を使用する前のテキスト

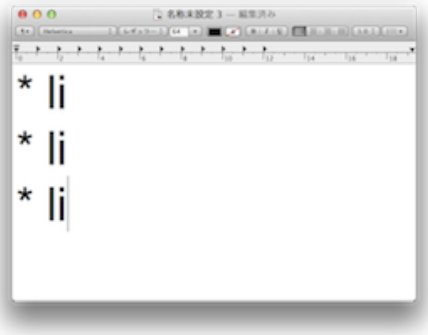

図 13. Dynamic Macro 機能を使用した後のテキスト

以下は SuprIME の Dynamic Macro 機能をブラウザのアドレスバー上で実行した例である．図 14 のように，ユーザの入力に”abc”が繰り返されているとき，任意に設定したファンクションキーを押すと，テキストは図 15 のように編集される．

図 14. Dynamic Macro 機能を使用する前のテキスト

図 15. Dynamic Macro 機能を使用した後のテキスト

Emacs 上に実装された Dynamic Macro は Emacs 上でしか利用できないが，SuprIME 上に実装された Dynamic Macro はあらゆるテキストエディタの上で利用することができる．

3 実装

SuprIME は MacRuby で記述された Mac 用の IME であり，ソースが 500 行程度とコンパクトであるにもかかわらず，他の IME に見られない機能を実装しており，本論文のような実験も容易である．

3.1 MacRuby による実装

MacRuby は，Mac OS 用のアプリケーションを開発するために拡張された Ruby 実行環境であり，Mac OS の Objective-C ライブラリを Ruby で扱うことが出来る．SuprIME では，Mac OS の IME フレームワークである InputMethodKit Framework を MacRuby から呼び出すことによって基本的な IME の機能を実装している．

3.2 修飾キーの扱い

エディタが編集機能のために用いる，Control や Command などの修飾キーは IME がハンドルしないのが普通であるが，SuprIME では、一部のファンクションキーと修飾キーの入力を受け取り，共通化されたテキスト編集操作を実現している．

SuprIME のように，修飾キーの一部をハンドルする実装を行った場合，修飾キーの入力はエディタに送出されないため，エディタが同じキー入力を編集操作の一部に割り当てている場合，それを利用することは出来なくなる．

3.3 テキストデータの取得

ブロック移動やインデントの処理といったテキスト編集の機能を，あらゆるテキストエリアで行うためには，テキストエリアに入力されている全文を IME が取得する必要があり，InputMethodKit はこの機能を備えていない．SuprIME では，ブロック移動やインデントのコマンドが入力された際に，Mac OS に実装されている，テキストエリアの全文を選択状態にするコマンドを AppleScript により送信している．

Mac OS では，多くのアプリケーションにおいてテキストエリアは NSTextField Class のオブジェクトとして実装されており，この NSTextField に対してテキスト編集や入力の操作を行うための NSTextInput Protocol が存在する．このプロトコルには標準の実装として Command+A キーによる全文選択の機能が備わっている．

SuprIME では，このキー入力を AppleScript によりテキストエリアに送信することにより，テキストエリアの全文を選択状態にしている．SuprIME では，NSTextInput の標準実装である Command+C キーによる全文コピーの機能を AppleScript により送信することで，Mac OS のクリップボードにテキストエリアの全文を保存し，それを InputMethodKit

Framework が読み込むことで，テキストエリアの全文を IME が取得している．

3.4 実装手法の問題点

しかし，このような手法で実装できる編集機能や，IME としてのパフォーマンスには問題がある．たとえば，対象とするテキストエリア自体が NSTextField Class を利用していない場合や，対象とするテキストエリアがテキストの全選択やコピーに割り当てるキーバインドを変更している場合，テキストエリアの全文を取得することが出来ない場合があり，SuprIME の持つ編集機能は有効に機能しない．また、どこまでの編集機能をエディタやテキストエリアから切り離し，どれだけの拡張性を持たせることができるか，という点において，IME の層での実装には問題がある．各国語入力のための枠組みである IME は，テキストエリアよりも先にユーザのキー入力をハンドルすることができるが，エディタが想定するキー入力を横取りして、エディタの機能をを打ち消す実装になる可能性がある．前述した Dynamic Macro の例のように、編集操作をマクロ化して行うことは多くあるので，IME とテキストエリアとの間に，テキスト編集のための枠組みを用意し，機能拡張が容易に可能な OS と API 設計のレベルでの実装の修正が求められる．

4 議論

4.1 キー操作の統一の必要性

Xerox PARC の Larry Tesler が 70 年代に発明した [10] 「コピー/ペースト」は現在広く普及しており，ほぼすべてのアプリケーションにおいて同じ操作 [3] でテキストをコピーしたりペーストしたりできるようになっている．つまりコピー/ペーストに関しては 1 章で述べたような問題がほぼ存在せず，ユーザは混乱せずに様々なアプリケーション上でテキスト操作を行なうことが可能になっていることになる．コピー/ペーストの操作がこのように標準化されているのに対し，テキスト移動のような処理が標準化されていないのは問題であるが，それが大きな問題だと認識されておらず，ユーザが様々なエディタの使い方を覚えることをシステムが強制していることはさらに大きな問題だといえるだろう．SuprIME のような手法を利用することによってシステム全体を統一的に利用できるようにする工夫はまだまだ重要だと考えられる．

4.2 ユニバーサルな入力/編集

パソコンやスマートフォンで利用されている様々なテキスト入力システムは変換方式も使い勝手も全く異なるのが普通になっているが，POBox[6] のような単純で柔軟な入力方式を利用すると，パソコンでもスマートフォンでもほぼ共通の入力を行なうことが可能になる．Gyaim はこのような思想にもとづいて作成された IME であるが，Gyaim を拡張した SuprIME を利用すると編集操作も共通化することができるため，広い範囲の機器における入力と編集の共通化が期待できる．

4.3 高度な文書編集補助

SuprIME を利用すると，単純な文書編集作業が共通化できるだけではなく，より高度な編集補助を行なうことも可能である．たとえば，単語の言い換えを補助するために，図 16 のように類語を検索して入力することが出来る．

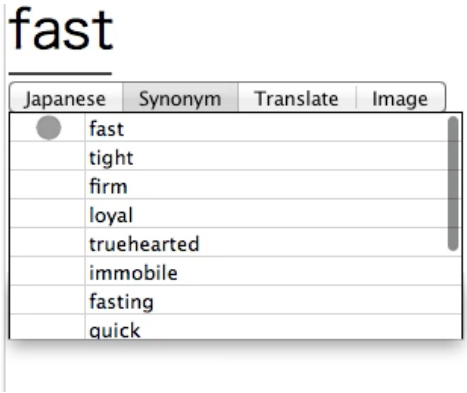

図 16. SuprIME の類語変換機能の例

図 17,18 は SuprIME の画像入力の例である．ここでは、「知らない」という入力語句について画像を検索し，入力候補として表示している．画像が埋め込めるテキストエリアであれば画像を埋め込み，そうでなければ URL が入力される．

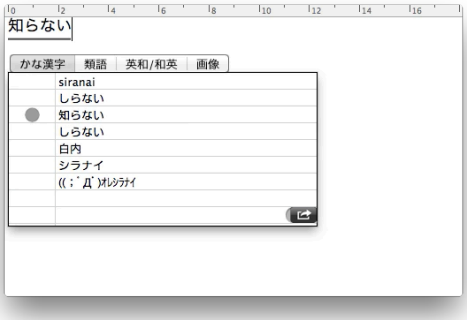

図 17. SuprIME で「知らない」という単語を入力した例

[3] Mac の場合は Command-C と Command-V

図 18. SuprIME の画像入力ウィンドウ

SuprIME を利用すると，このような高度なテキスト編者操作もあらゆるアプリケーションで共通に利用できる．

4.4 テキスト入力/編集以外への応用

本論文ではテキストエディタに絞った説明を行なったが，IME はユーザのすべてのキー入力を直接受け取る窓口になっているため，キー操作をテキスト編集と関係無い仕事に割り当てることも可能である．たとえばシステム音量や画面の明るさなども SuprIME から制御することが可能であり，システムで用意されたショートカット設定などが不要になる．

4.5 本手法の限界

IME は，テキストエリアが実装されたアプリケーションとは個別に実装されているため，現在の SuprIME の実装ではアプリケーションの内部状態によって動作を変えたり，アプリケーションの振る舞いを制御したりすることはできず，表に出ているテキストの編集操作しかできない．

既存のシステム自体は変更せずに，皮をかぶせる形で機能を拡張する手法はある程度有用ではあるが，問題の根本的な解決が必要な場合には限界がある．テキスト編集の場合は根本的に解決しなければならない問題は多くないので，本論文の手法はとりあえず有効だといえるが，根本的な解決のためには，テキスト入力の枠組みである IME に加えて，各コンピュータがテキスト編集操作を一元化するための枠組みを用意する必要があるだろう．

5 結論

OS に標準装備された IME 機能を活用することにより，様々なテキストエディタにおける入力/編集操作を共通化する手法を提案した．IME 機能はアプリケーションから独立しているため，アプリケーション内のデータを操作することはできないが，様々なアプリケーションにおける入力/編集操作の多くの部分を共通化することができた．テキストの入力や編集は計算機利用における最も重要な仕事のひとつであることは間違いないので，操作を簡単化/共通化するための優れた枠組みの開発は重要な課題だと考えられる．

参考文献

[1] TextEditors.org. http://texteditors.org/.

[2] P. H. Dietz, B. Eidelson, J. Westhues, and S. Bathiche. A practical pressure sensitive computer keyboard. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, UIST '09, pp. 55–58, 2009.

[3] C. Harrison, D. Tan, and D. Morris. Skinput: appropriating the body as an input surface. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pp. 453–462, 2010.

[4] F. C. Y. Li, R. T. Guy, K. Yatani, and K. N. Truong. The 1line keyboard: a QWERTY layout in a single line. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, pp. 461–470, 2011.

[5] I. S. MacKenzie, R. W. Soukoreff, and J. Helga. 1 thumb, 4 buttons, 20 words per minute: design and evaluation of H4-writer. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, pp. 471–480, 2011.

[6] T. Masui. An efficient text input method for pen-based computers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '98, pp. 328–335. ACM Press/Addison-Wesley Publishing Co., 1998.

[7] T. Masui and K. Nakayama. Repeat and predict – two keys to efficient text editing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '94, pp. 118–123, 1994.

[8] T. Murase, A. Moteki, N. Ozawa, N. Hara, T. Nakai, and K. Fujimoto. Gesture keyboard requiring only one camera. In *Proceedings of the 24th annual ACM symposium adjunct on User interface software and technology*, UIST '11 Adjunct, pp. 9–10, 2011.

[9] J. Rick. Performance optimizations of virtual keyboards for stroke-based text entry on a touch-based tabletop. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, UIST '10, pp. 77–86, 2010.

[10] L. Tesler. A personal history of modeless text editing and cut/copy-paste. *interactions*, 19(4):70–75, July 2012.

[11] D. Wigdor and R. Balakrishnan. TiltText: using tilt for text input to mobile phones. In *Proceedings of the 16th annual ACM symposium on User interface software and technology*, UIST '03, pp. 81–90, 2003.