CHAPTER

Ten

Composition by Example

TOSHIYUKI MASUI

Sony Computer Science Laboratories, Inc.

S
R
L

Abstract

Programming by example (PBE) literally means making "programs" from examples, but PBE techniques are also useful for automating text composition tasks by creating editing procedures from the history of a user's editing operations. *Composition by example* (CBE) is a practical approach to improve the efficiency of text composition tasks using various simple PBE techniques. Thousands of people are composing texts daily on handheld computers and Emacs using CBE tools that I have distributed for years.

10.1 Introduction

Although text composition is a highly creative task, it can be full of repetitive menial subtasks. For example, people sometimes compose letters by duplicating an old letter whose topic is close to the new one. They repeat the same search-replace operations or type the same phrases (e.g., their names and addresses) again and again in different documents. The popularity of small portable text devices is now presenting users with new classes of repetitive text composition subtasks. On cellular phones, for instance, the user must type small keys many times when composing messages. If a system could automatically perform these repetitive tasks based on examples given implicitly by the user, people could compose texts much more efficiently.

To automate text composition tasks, many text editors support end-user programming features so that repetitive tasks can be programmed by users. For example, on GNU Emacs, users can define and execute arbitrary functions by writing Emacs Lisp codes. Users of Emacs can also define a sequence of keystrokes as a macro command and later invoke the sequence by a single keystroke. With these features, any repetitive task can be defined as a function or a macro.

However, writing a program or defining a macro is not always a convenient way for performing repetitive tasks, even for trained programmers. If a user has to enter a symbol at the beginning of 5 consecutive lines of a text file, he would use a text editor and perform the task manually. If he has to do the same thing for 10,000 lines, he would write a program. But what if he has to do it for 100 lines? Is it worthwhile writing a program?

In this chapter, I show that simple example-based techniques can make various text composition tasks much easier. The first example is the *Dynamic Macro* system (Masui and Nakayama 1994), which enables users of

text editors to repeat arbitrary editing operations after executing the same operations more than once. The second example is the *POBox* system (Masui 1999), which enables efficient text input by predicting the next input word from the context, using a dictionary created by the user's text input histories. Using these CBE techniques, users can efficiently create and edit texts without noticing that they are giving examples and making programs from them.

Dynamic Macro and POBox are not paper systems but are actually being used by thousands of people. Dynamic Macro has been very popular in the Emacs user community for several years. I have been using POBox for all my Japanese composition on handheld computers and Emacs. The Japanese version of POBox on Palm Pilot has been available on the Web for more than two years, has been updated many times after receiving suggestions and bug reports from the users, and is currently used by thousands of people. POBox has been adopted as the official text input method of Sony's cellular phones and information appliances, which expect millions of users.

10.2 PBE-Based Text Editing Systems

Many researchers noticed that text-editing tasks involved a lot of menial routine works and tried to improve the efficiency by various PBE techniques. In Nix's (1985) Editing by Example system, users can tell the system to infer the editing procedure by showing the text both before and after modification. The inferred procedure should be of the "gap programming" form, which is a subset of string substitutions using regular expressions. Mo and Witten's (1992) TELS system generalizes users' iterative operations and infers an editing procedure including loops and conditional branches. If the system's guess is wrong, users can incrementally correct it until it does the right thing for them. Since the procedure generated by TELS can include branches and loops, it can perform complex tasks that cannot be done by mere string substitutions. GNU Emacs provides the "dabbrev" function, which expands the substring entered by the user into a full string in the same document beginning with the same substring.

10.3 Dynamic Macro: A PBE-Based Text Editing System

Dynamic Macro is a simple and powerful tool for automatically creating a keyboard macro from repetitive user operations in a text editor. In many

text editors, a keyboard macro is used to substitute a long sequence of operations by another single operation. It is usually defined through the following steps: First, the user tells the editor to start recording a keyboard macro; second, she types the sequence of commands that she wants to define as a new macro; and finally, she tells the editor to stop the recording. For example, if a user of GNU Emacs wants to define a macro to insert a "%" at the top of every line, he types "^X (" to start the recording; then he types "^A % ^N" to insert a "%" at the top of the current line and go to the next line; then he types "^X)" to finish the recording. After the recording is finished, he can invoke these operations by typing "^X e". Although keyboard macros are general and powerful tools for repetitive editing tasks, they have several disadvantages. First, users have to remember three commands to record and invoke a keyboard macro. Second, it is not possible to define the command sequence after they are executed: that is, a user should know that a sequence of commands is used many times, well *before* actually executing them. In reality, repetitive tasks are often recognized *after* execution. Third, since the procedure of defining a keyboard macro is not simple, it is not useful for short, small repetitive operations.

Using Dynamic Macro, keyboard macros for repetitive operations are defined and executed automatically. When a user hits a special "REPEAT" key after performing repetitive operations, an editing sequence corresponding to one iteration is detected, defined as a macro, and executed at the same time. Although simple, a wide range of repetitive tasks can be performed just by hitting REPEAT.

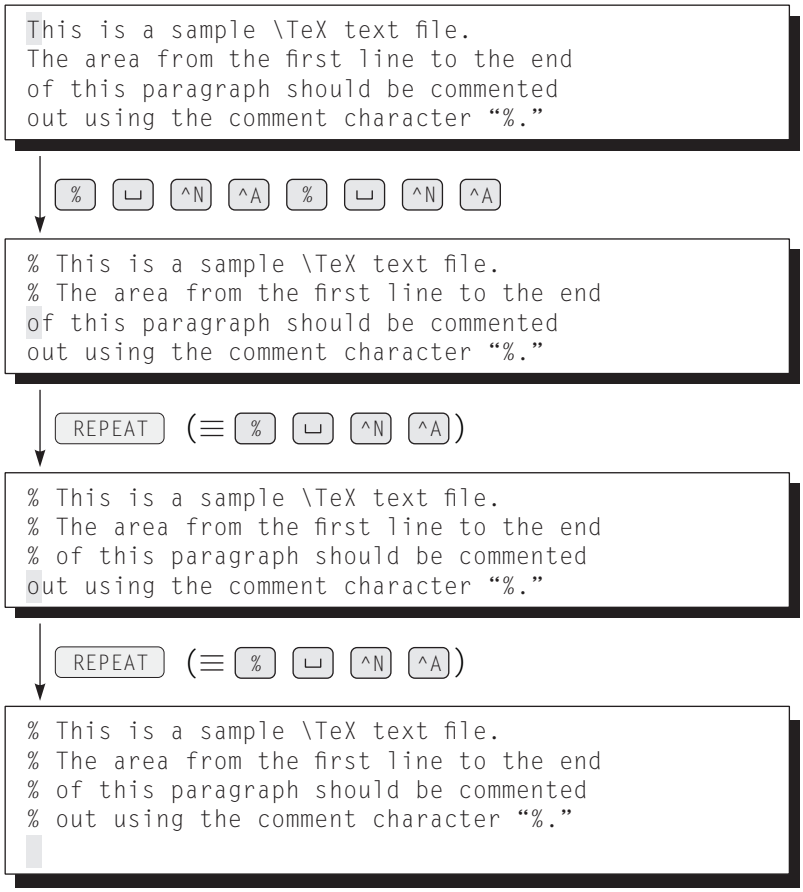
Dynamic Macro works as follows: All the recent user operations in a text editor are logged as a string, and when a special repeat command is issued by hitting REPEAT, the system looks for repetitive operations from the end of the string. If such operations are found, they are defined as a macro and executed. If REPEAT is struck again, the macro is executed again. For example, when a user enters the string abcabc and then hits REPEAT, the system detects the repetition of abc, defines it as a macro, and executes the macro, resulting in another abc. When the user hits REPEAT again, one more abc is inserted.

Similarly, when a user inserts a "%" at the top of two lines by doing the same operations twice and hits REPEAT, the operations are defined as a macro and executed. As a result, another "%" is inserted at the top of the third line.

Dynamic Macro does not suffer from the shortcomings of keyboard macros. Users should only remember that striking REPEAT makes the system do the repetitive task once more, instead of remembering three different operations of a keyboard macro. The macro is defined after doing

S
R
L

FIGURE **10.1**

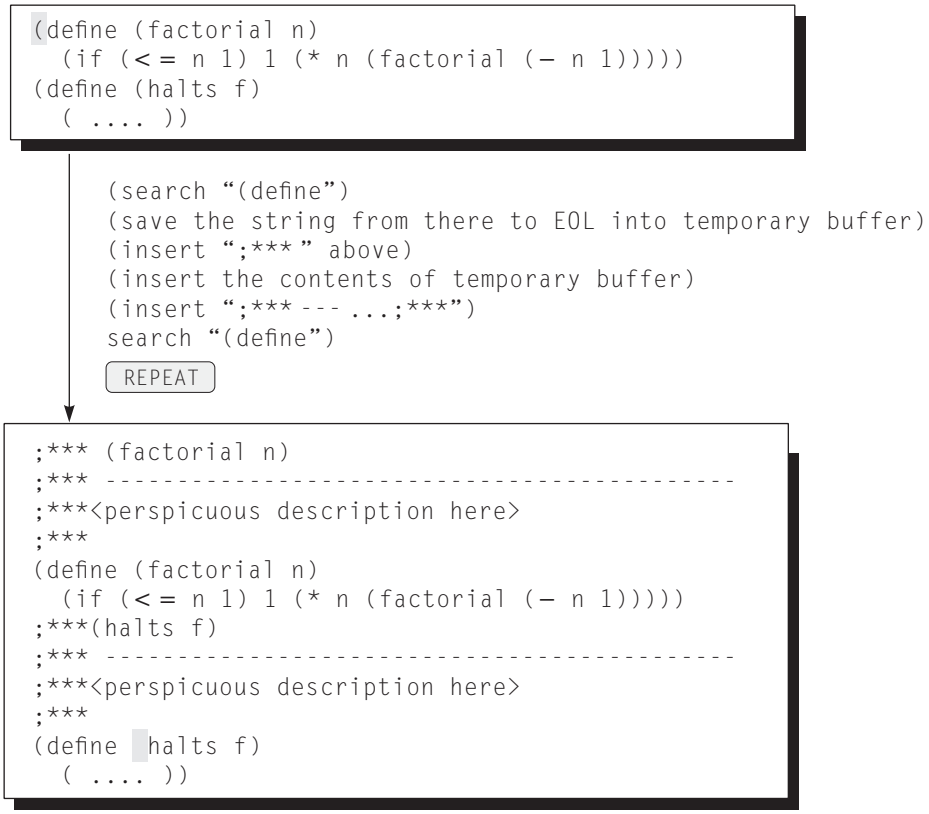


Adding comment characters at the beginning of each line in Dynamic Macro.

ordinary editing tasks, without telling the editor when to start the recording. In spite of its simple looking appearance, Dynamic Macro is applicable to various editing situations.

Figure 10.1 shows how Dynamic Macro works for simple tasks such as adding comment characters to consecutive lines. Figure 10.1(a) shows the original text. When a user types "% U ^N ^A % U ^N ^A", she gets Figure 10.1(b). If she hits REPEAT here, the system detects the repetition of "% U ^N ^A", defines the sequence as a macro, executes the macro, and gets Figure 10.1(c). Hitting REPEAT again results in Figure 10.1(d).

FIGURE **10.2**



Adding comment lines before each function definition.

Figure 10.2 shows a more complicated example. The job here is to add several comment lines above every function definition of Figure 10.2(a). This is done by long steps of operations, but striking REPEAT after doing the first part of the second iteration results in Figure 10.2(b). More hits will add similar comment lines to the following function definitions. In this case, since no sequence of operations is executed twice before REPEAT, the system searches the pattern XYX and defines XY as the macro.

The advantages of Dynamic Macro are as follows. First, it is *simple to use*. Users only have to remember that they can hit REPEAT to make the system do their repetitive chore. They can strike REPEAT at any moment during the repetitive operations, and they do not have to tell the system to start

or stop recordings. Second, it is *powerful*. Dynamic macro works well for a variety of simple to complex repetitive tasks where once only keyboard macros were applicable. Third, it is *easily implemented*. The system just keeps a log of recent user actions for the implementation of Dynamic Macro. Fourth, it *does not interfere with users* in any sense. Logging user actions is an easy task for most systems, and it does not slow down the application. Nothing happens unless users touch REPEAT. Finally, it is *general* in that any system with a keyboard interface can adopt this technique.

10.4 POBox: A PBE-Based Text Input System

Although millions of handheld computers and mobile phones are used today, and email and short messages are exchanged throughout the world, handheld and wearable computing have not really taken off, partly because of the lack of efficient text input methods. POBox is an example-based text input method especially effective for handheld and wearable computers where a full-size keyboard cannot be used and fast text input is difficult.

10.4.1 Various Text Input Techniques

Traditionally, on pen-based handheld computers, handwriting recognition techniques and the soft keyboard (i.e., the virtual keyboard displayed on the tablet of a pen computer) used to be the main techniques for entering text, along with others. However, using any of these techniques takes much longer to enter text than with a standard keyboard.

The situation is worse for East Asian languages such as Chinese and Japanese. Unlike European languages, these have thousands of character faces. Even with a keyboard, it is not easy to enter a character. A variety of techniques for entering text into computer have been investigated. The most widely used Japanese input technique is Roman-Kanji conversion (RKC), in which a user specifies the pronunciation of a word with an ASCII keyboard, and the system shows the user a word with the specified pronunciation. If the word was not the one that the user intended to use, the user hits a "next-candidate key" until the correct word appears as the candidate.

Figures 10.3 shows an overview of various existing text input systems. Arrow A shows how an English text is composed using a standard keyboard. Roman character codes are directly generated by the keyboard and concatenated to generate a text. Arrow B shows how a Japanese text is

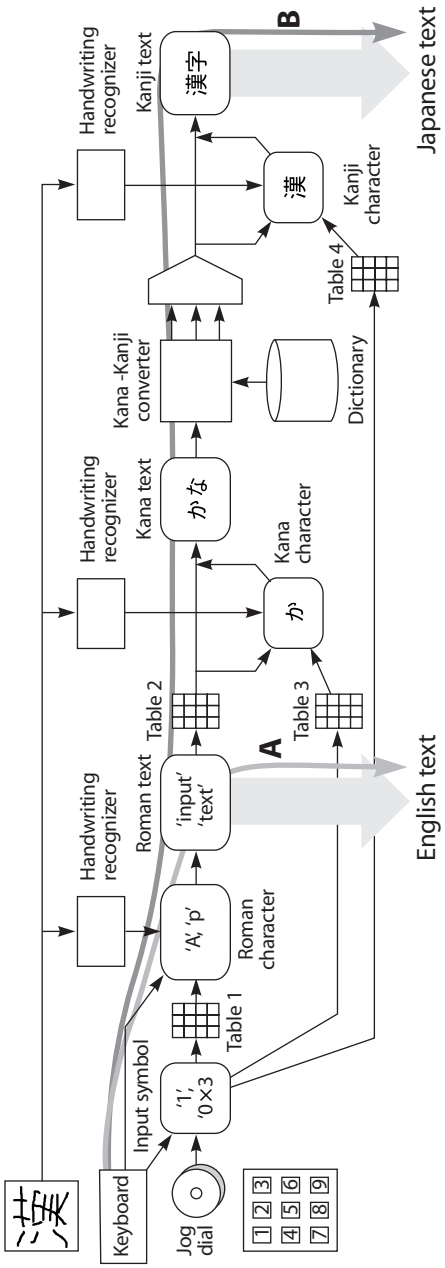


FIGURE 10.3

Structure of various text input methods.

composed using a standard keyboard. Roman character strings are first converted to Kana texts that represent the pronunciation of Japanese words, and then they are converted to Kanji characters by a Kana-Kanji converter. Since multiple Kanji characters often have the same pronunciation, the user must choose the correct one with the selector.

Text input on handheld computers is very slow for many reasons. First, typing a key or writing a character is much slower than using a standard keyboard. Second, users have to type keys more times than when using standard keyboards, since small input devices often have fewer keys (e.g., cellular phones usually have only 20 keys). These keys can generate only a small number of input symbols, and combinations of the keypress must be converted to Roman characters using a mapping table. In this way, input symbols must be converted more than once until the final text is composed.

Entering Japanese text on a cellular phone is also very slow. The input symbols must first be converted to Kana characters using Table 3 in Figure 10.3, and then a Kana character string is converted to a Kanji character using the Kana-Kanji converter. A proper Kanji must then be selected using the selector.

Several techniques for fast text input on handheld machines have been proposed. One approach is to make the speed of using a software keyboard faster. The QWERTY layout is often used for the software keyboard, but QWERTY is not the best layout for a pen-based software keyboard, since frequently used key combinations are sometimes laid out far apart and users must move the pen for a long distance to enter a text. The Fitaly keyboard[1] is a layout for minimizing the pen movement on software keyboards. Since "e" and "n" often appear next to each other in many English words, they are put in an adjacent position on the Fitaly keyboard. Other layouts have also been proposed to improve the input speed on software keyboards (see Hashimoto and Togashi 1995; MacKenzie and Zhang 1999).

Another approach is to use fast handwriting recognition systems. Unistroke (Goldberg and Richardson 1993) was one of the first approaches in this direction, and similar techniques such as Graffiti have become very popular on recent handheld computers including Palm Pilot. More sophisticated gesture-based techniques such as T-Cube (Venolia and Neiberg 1994), Quikwriting (Perlin 1998), and Cirrin (Mankoff and Abowd 1998) have also been proposed.

Yet another approach is to give up entering characters one by one and to use a word dictionary for composing a text. Textware's InstantText system

1. Textware Solutions, 83 Cambridge St., Burlington, Mass., 01803. *www.twsolutions.com/fitaly/fitaly.htm.*

S
R
L

(see footnote 1) allows users to use an abbreviated notation of a sentence to reduce the number of input. For example, users can type `oot` to enter "one of the" or type `chrtcs` to enter "characteristics". These abbreviations are dynamically created, and they do not have to be predefined.

Tegic's T9 system[2] takes a different approach. T9 was originally developed for composing texts using only nine keys on a standard telephone. On T9, instead of typing keys more than once to select an input character, users assign more than one character to the digit keys of a telephone so that they do not have to be concerned about the differences. Figure 10.14 on page XXX shows a typical key assignment on a telephone keypad. When a user wants to enter is, he pushes the `4` key first, where `G`, `H`, and `I` are printed, and then pushes the `7` key where `P`, `Q`, `R`, and `S` are printed. Using the combination of `4` and `7` corresponds to various two-character combinations, including `hr`, `gs`, and so on, but `is` appears most frequently in English texts, and the system guesses that is is the intended word in this case.

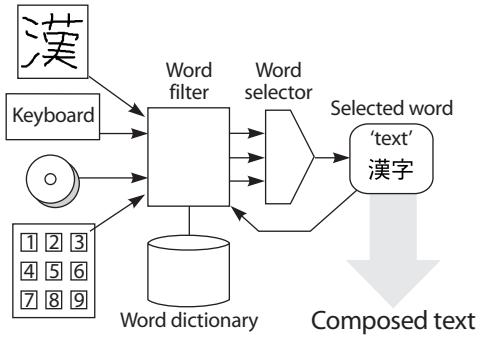
On almost all the pen-based computers available in Japan, either RKC or handwriting recognition is supported. Text input is slow and tiring using either of the techniques for the following reasons. Specifying the pronunciation of every input word using a soft keyboard takes a lot of time, and the user must convert the pronunciation to the desired Kanji strings with extra keystrokes. Handwriting recognition has more problems. First, the recognizer has to distinguish between thousands of characters, often making errors. Many of the characters in the character sets have similar shapes, so it is inherently difficult to make recognition reliable. Second, in many cases, users do not remember the shape or the stroke order of Kanji characters, even when they have no problem reading them. Finally, writing many characters with many strokes on a tablet is very tiring. With these difficulties, it is believed to be difficult to enter Japanese text faster than thirty characters per minute on pen-based computers, which is several times slower than using keyboards.

10.4.2 POBox Architecture

Since people usually compose texts that are in some way close to old texts, an example-based approach can be applied to solve this problem. Using POBox, text is not composed by entering characters one by one, but by selecting words or phrases from a menu of candidates created by filtering the dictionary and predicting from context. Word dictionary and phrase dictionary are first created from existing corpus, and updated by examples S R L

2. Tegic Communications, 2001 Western Ave., Suite 250, Seattle, Wash. 98121. *www.t9.com.*

FIGURE **10.4**



A POBox architecture.

given by the user. With this example-based approach, users can enter text much faster than recognition-based and other existing text input methods. Figure 10.4 shows the architecture of POBox. A text composition task with POBox consists of repetitions of the following two steps.

1. *Filtering step:* First, a user provides prediction keys for a word she wants to enter. Prediction keys can be the spelling, pronunciation, or shape of a character. As soon as she enters prediction keys, the system dynamically uses the keys to look for the word in the dictionary and shows candidate words to the user for selection.
2. *Selection step:* Next, the user selects a word from the candidate list, and the word is placed in the composed text. The selected word and the current context are saved in the dictionary as a new example and used in the future filtering step, so that the word is properly picked up as a candidate in the same context next time.

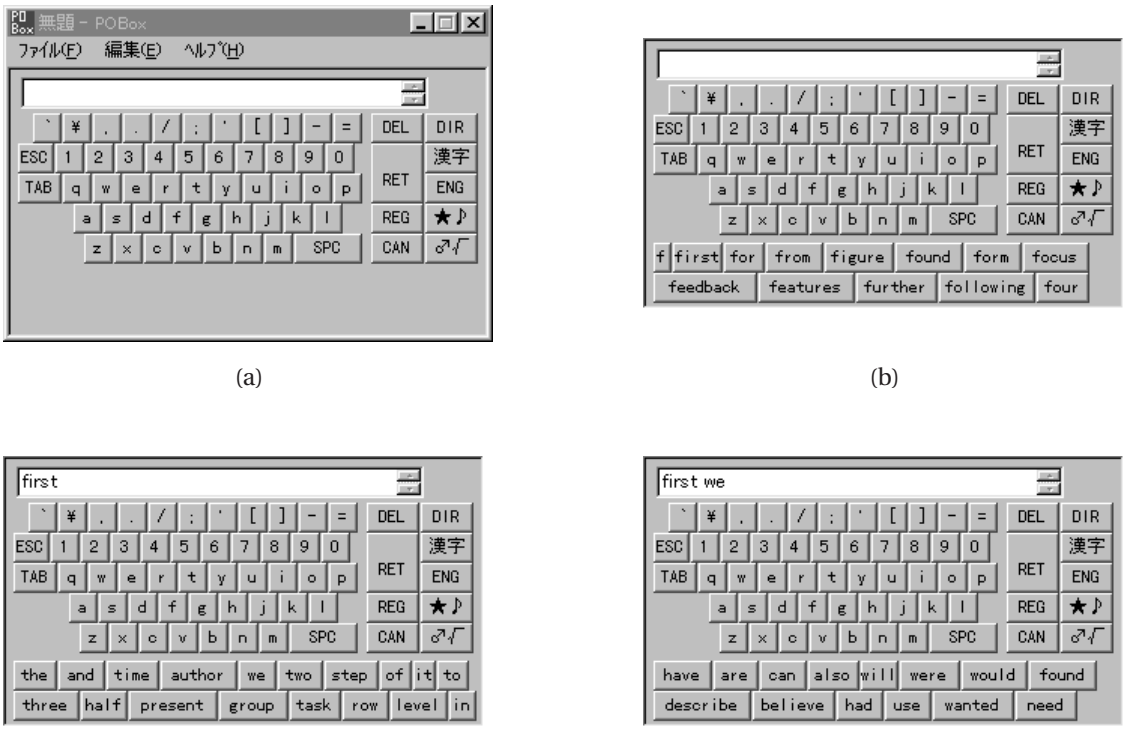
In most existing text input systems, users must provide all the information for the input text, either by specifying input characters or by showing the complete shape of characters by giving handwritten strokes. In POBox, users do not have to give all of them to the system; they only have to give information to the system that is enough for the prediction. Users also do not have to specify all the characters or stroke elements that constitute a word; they only have to specify part of the input word and select it from the candidate list. This greatly reduces the amount of operations and time for composing a text, especially when selecting input characters is very slow or

difficult. This architecture can be applied to a variety of nonkeyboard devices, including pen tablets, one-hand keyboards, and jog-based phones.

10.4.3 POBox for Pen-Based Computers

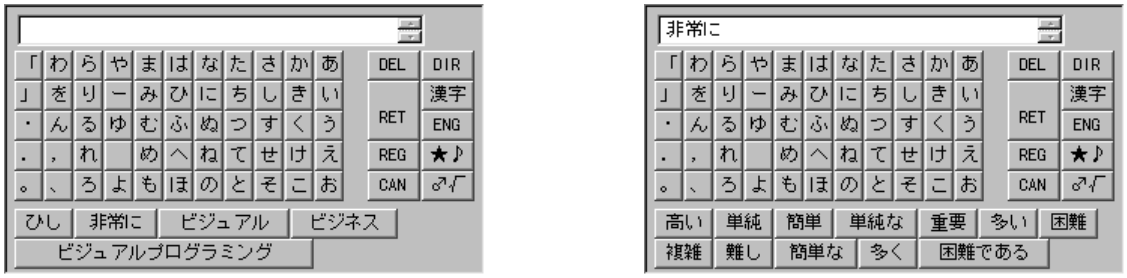
Figure 10.5(a) shows the startup display of POBox running on Windows95. When the user pushes the `F` key, the display changes to Figure 10.5(b), showing the frequently used words that start with `F` in a candidate word list. Since the word `first` is a frequently used word and is found in the candidate list, the user can tap the word `first` so that it is put into the text area.

FIGURE 10.5



POBox running on Windows95: (a) POBox's startup display; (b) frequently used words starting with `f`; (c) words that often follow `first`; and (d) user-chosen word order of first we.

FIGURE 10.6



(a) (b)

Japanese input mode of POBox: (a) a Hiragana character table and (b) a user-selected word from Hiragana display.

After first has been selected, the display changes to Figure 10.5(c). In the menu at the bottom, the words that often come after first are listed in order of frequency.

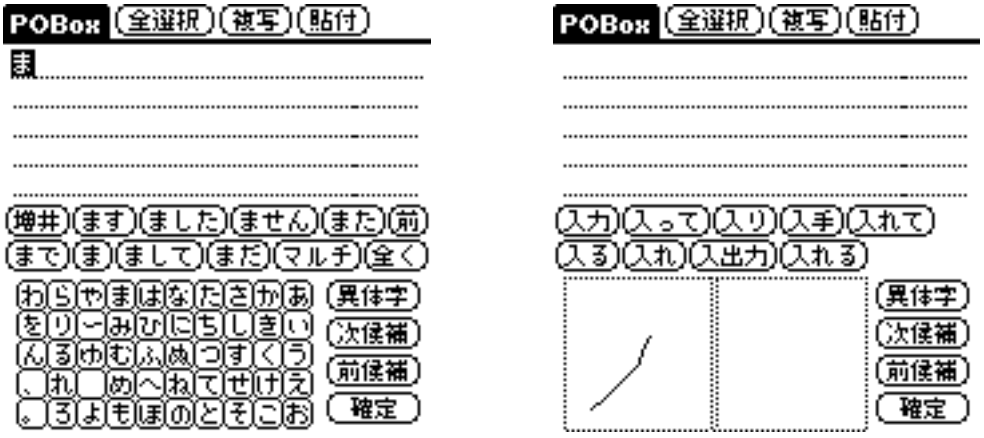
The next word, we, often comes after first, and this word is again in the predicted list of candidate words. The user can directly select we by touching it in the menu. After we has been selected, the display changes to Figure 10.5(d). In this way, users can repeatedly specify the prediction key and select a candidate word to compose a text.

In the Japanese input mode of POBox, a Hiragana character table is displayed for entering pronunciations, instead of the Roman alphabet in English mode. The pronunciation of the first word 非常に [is hi-jou-ni, and the user can select the word by choosing word ひ (hi) and word し (shi) from the Hiragana keyboard, just like in the English example (Fig. 10.6[a]).

When the user selects word 非常に, the display changes to Figure 10.6(b), and the next word word 単純な is displayed as candidate in the candidate word list at the bottom. In this way, the user can enter Japanese text by specifying the pronunciation of the first portion of the word and then selecting the desired word from the menu, just like specifying the spelling for English words.

Figure 10.7(a) shows the display of POBox on Palm Pilot after the user hits the ma key on the software keyboard. The words listed are candidate words beginning with the pronunciation ma. When the user moves the pen after touching the tablet instead of tapping the software keyboard, the system starts handwriting recognition and interprets the strokes incrementally, showing candidate words that begin with the strokes. Figure

FIGURE 10.7



(a) (b)

The display of POBox on Palm Pilot (a) after the user hits the ma key and (b) after the user has drawn a line from the center of the keyboard to the lower-left corner.

10.7(b) shows the display after the user has drawn a line from the center of the software keyboard to the lower-left corner. This is the first stroke of the Kanji character "nyuu", and those words that begin with the character are shown as candidates. Unlike existing handwriting recognition systems that recognize characters only after all penstrokes that constitute the character have been written, incremental recognition can greatly reduce the number of penstrokes that users have to draw. In this way, software keyboards and handwriting recognition are seamlessly integrated in POBox for pen-based computers.

10.4.4 Using POBox on a Cellular Phone

POBox can be used for handheld devices that do not have pen tablets. Instead of using a software keyboard or pen operations on a cellular phone, digit keys and a jog dial can be used for the filtering and the selection steps.

Figure 10.8(a) shows the implementation of POBox on a CDMA (code-division multiple access) cellular phone. The phone has about twenty keys on the surface and a jog dial at the left side of the LCD display. Three or four alphabetical characters are assigned to each digit key (Figure 10.8[b]) like standard push phones in North America. Hiragana characters are also

S
R
L

FIGURE 10.8

(a) (b)

(a) The implementation of POBox on a CDMA cellular phone and (b) a display of the digit key containing alphabetical characters.

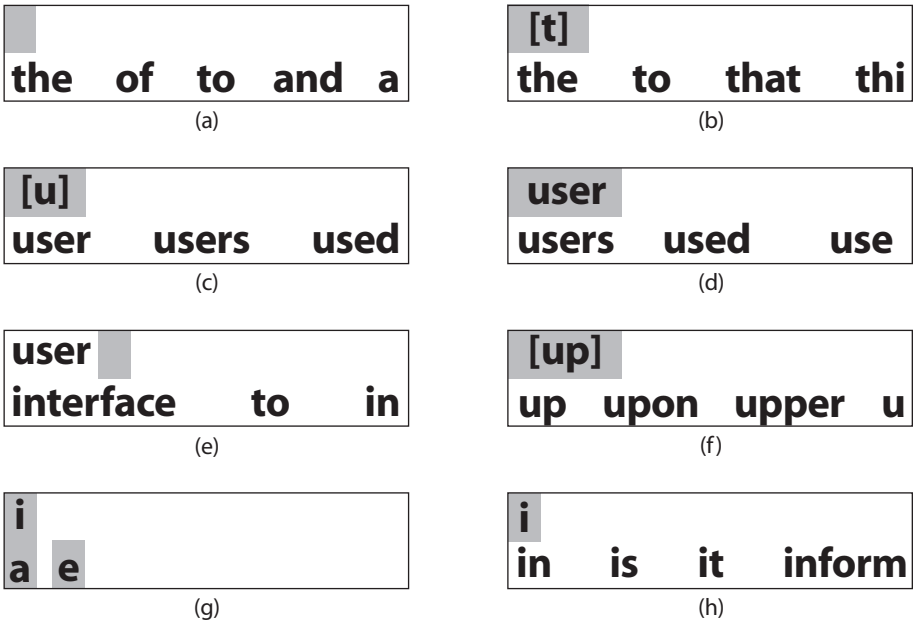
assigned to those keys to specify the pronunciation when used for Japanese text input.

Figure 10.9(a) shows the initial display of the phone. Frequently used words are listed as candidates at the bottom of the display. When a user pushes one of the digit keys, the character printed on the key is shown at the cursor position, and candidate words starting with the character are shown at the bottom of the display (Figure. 10.9[b]). When the user pushes the key again, the next character printed on the keytop is shown, and corresponding candidate words are displayed (Figure. 10.9[c]). A user can rotate the jog dial clockwise at any time to select a candidate word. If user is the desired word, the user can rotate the jog dial and display user at the top of the display. As the user changes the selection, more candidate words appear at the bottom of the screen for selection (Figure. 10.9[d]).

The user can then push the jog dial to make the selection final. At this moment, the next word is predicted just like in pen-based POBox, and next candidate words are displayed at the bottom. The user can again rotate the jog dial to select a candidate from the list (Figure. 10.9[e]). In Figure 10.9(c), if the user pushes the 7 key, `p` is selected as the next character for the input word (Figure. 10.9[f]). When a user begins rotating the jog dial counterclockwise, she can select input characters by the jog rotation. Input characters are sorted in frequency order; `e`, `a`, `i`, and so forth appear as the candidate input character as the user rotates the jog dial. Figure 10.9(g) shows the display after the user rotated the dial three steps.

S
R
L

FIGURE 10.9



Text input steps on a CDMA phone: (a) initial display; (b) after pushing the 8 key; (c) pushing the 8 key again; (d) rotating the jog dial clockwise; (e) pushing the jog; (f) pushing the 7 key after (c); rotating the jog dial counterclockwise; and (h) pushing the jog.

When the user pushes the jog dial, the search character becomes fixed, and words that begin with the pattern are displayed as candidates (Figure. 10.9[h]). The user can then rotate the jog dial clockwise to select the candidate input word (e.g., `information`). Although using a jog dial for character input takes more time than using digit keys, using a jog dial has an advantage; users do not have to touch the digit key at all, so composing text using only one hand is possible.

10.4.5 POBox Server on the Internet

If POBox can be implemented as a server, various client computers can connect to it and ask the server to do the prediction. In this case, users can use the same system from any terminal, and all the example words and

phrases are saved in the server for later prediction. For example, when a user enters a proper noun on a handheld computer, the name can be used for prediction on a desktop computer. If everyone has his or her personal POBox server on the Internet, people would no longer have to enter difficult spelling more than once.

10.4 Conclusion

This chapter introduced two very practical PBE-based tools for text composition. Some people argue that PBE is sometimes not as effective as it is expected to be. However, text composition is one of the areas where PBE-based technique is truly effective. I use Dynamic Macro almost every day and POBox for all the Japanese documents on Emacs and my Palm Pilot, getting rid of all other Japanese text input methods. POBox for Palm Pilot has been on the Web for more than two years, and tens of thousands of people have downloaded it, since it is the fastest Japanese text input method on handheld computers.

I believe that PBE techniques are most effective in the following two cases: when very simple prediction is enough for the task, such as the case of Dynamic Macro and POBox, and when the task is very complicated and even human programmers cannot easily create programs to solve the problem. In this case, stocastic methods for creating programs from examples are effective. Graph layout tasks and other aesthetic tasks are in this category. In short, PBE is effective for a variety of composition tasks that entail both highly creative aspects and routine work. I hope that more PBE-based techniques are investigated for various composition tasks.

References

Goldberg, D., and C. Richardson. 1993. Touch-typing with a stylus. In *Proceedings of ACM INTERCHI '93 Conference on Human Factors in Computing Systems (CHI '93)*, April 1993. Reading, Mass.: Addison-Wesley.

Hashimoto, M., and M. Togasi. 1995. A virtual oval keyboard and a vector input method for pen-based character input. In *CHI '95 Conference Companion*, May 1995. Reading, Mass.: Addison-Wesley.

MacKenzie, I. S., and S.-Z. Zhang. 1999. The design and evaluation of a high performance soft keyboard. *In Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '99),* May 1999. Reading, Mass.: Addison-Wesley.

Mankoff, J., and G. D. Abowd. 1998. A word-level unistroke keyboard for pen input. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '98),* November 1998. ACM Press: http://mrl.nyu.edu/perlin/demos/quikwriting.html.

Masui, Toshiyuki. 1999. POBox: An efficient text input method for handheld and ubiquitous computers. In *Proceedings of the International Symposium on Handheld and Ubiquitous Computing (HUC '99),* September 1999.

Masui, Toshiyuki, and Ken Nakayama. 1994. Repeat and predict—Two keys to efficient text editing. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '94),* April 1994. Reading, Mass.: Addison-Wesley.

Mo, D. H., and I. H. Witten. 1992. Learning text editing tasks from examples: A procedural approach. *Behaviour & Information Technology* 11, no. 1: 32–45.

Nix, R. P. 1985. Editing by example. *ACM Transactions on Programming Languages and Systems 7,* no. 4 (October): 600–621.

Perlin, K. 1998. Quikwriting: Continuous stylus-based text entry. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '98),* November 1998. ACM Press: http://mrl.nyu.edu/perlin/demos/quikwriting.html.

Venolia, D., and F. Neiberg. 1994. A fast, self-disclosing pen-based alphabet. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '94),* April 1994. Reading, Mass.: Addison-Wesley.

S
R
L