EDITING BY VOICE AND THE ROLE OF SEQUENTIAL SYMBOL SYSTEMS FOR IMPROVED HUMAN-TO-COMPUTER INFORMATION RATES

Nils Klarlund

AT&T Labs–Research, 180 Park Avenue, Florham Park, NJ 07932

ABSTRACT

Composing text on the computer is usually heavily dependent on editing, such as moving text around, correcting spacing, and inserting punctuation characters. Dictation systems, based on automatic speech recognition, are not known for their efficiency as an editing tool—something that significantly reduces their potential for freeing the user from the keyboard.

Speech recognition has almost invariably been tied to natural language, but we point out that this approach is inherently disadvantageous in important ways. Instead, given the evidence that humans routinely become experts at sequencing signs not related to natural language, we propose that editing command languages should rely on symbolizations similar to that of the keyboard.

We introduce ShortTalk, an editing language whose command sequences are primitive symbol combinations that are not confusable with dictation. We argue that ShortTalk by construction may solve common editing situations much more efficiently than by use of keyboard and mouse.

Our experimental results for ShortTalk indicate that an average information rate of about 16 bps for editing commands is achievable. Thus, editing by speech may be more efficient than by non-verbal means.

1. INTRODUCTION

The physical strain of using computers is significant[1]. But for the apparently large number of computer users suffering from CTD (Cumulative Trauma Disorder such as tendinitis of the forearms) or other soft tissue pain (carpal tunnel syndrome, neck pain, etc.), dictation systems have largely been a disappointment according to anecdotal evidence (and documented, spectacular business failures).

Thus, there is a need for principled efforts at designing speech-operated tools that interface with the human mind as naturally and as efficiently as the keyboard.

But, for laypeople and researchers alike, speech recognition is linked to natural language. Indeed, the orthodox view of the role of voice input is that users may expect to engage in a dialogue with the machine using ordinary linguistic constructs, see [2]. For many applications, such as for call processing, see [3], there is indeed no alternative, since users are assumed to be unskilled. In addition, there is a semantic significance to natural language, because natural language is what is used when a caller converses with a human agent.

In this paper, we argue on the contrary that for spoken command and control there is little communicative significance to natural language and that humans are best served with primitive systems of sequentially combined symbols. Our belief is that efficiency of user interfaces departs from an understanding of the human ability to learn and sequence symbols with ordained technical meanings.

We formulate two principles for realizing this goal for dictation systems: *stenophonic concept naming*—fundamental concepts are treated as symbols that receive short pronunciations—and *unambiguous orthogonality*—the command grammar must allow concepts to be sequenced combinatorially with few restrictions while preserving distinctness of commands from fragments of natural language.

Our techniques are exemplified through the artificial language ShortTalk, which has been developed over a six-year period to be a comprehensive and universally effective tool for text manipulation, whether the domain is e-mail writing, XML content authoring, programming, etc. We provide a summary of ShortTalk that illustrate the considerable expressive power achievable by our techniques.

We report on a long-term experiment, which has shown that the average command entropy rate of ShortTalk may reach 16bps. Taken together, our cognitive arguments, analysis, and results indicate that current speech user interfaces may be dramatically improved if overlooked human capacities for acquiring and sequencing symbols are acknowledged. The success of Graffiti, a simplified and artificial representation of letters, should inspire research in dictation systems, where user interfaces lag the impressive advances in the speech recognition technology itself.

1.1. Editing as symbol sequencing

Areas of automation where speech recognition has been applied include form filling on PDAs, operating household appliances, or information retrieval over the phone. Intrinsically different from these domains, text editing is a manifestation of the trained mind and its task-oriented information-processing facilities. And, importantly, there is no natural efficient way to talk about editing, neither to a human nor to a computer. Indeed, most of us have experienced jointly editing a paper in front of the computer screen, where we feel frustrated from the inadequacy of our language to convey our editing intentions. A graphical mixture of proofreader's marks, lines, and boxes seems to be the most efficient vocabulary for expressing changes to a text.

Innumerable text editors have addressed the very same problem through different modalities. Later ones have used the graphical abilities provided by pointing devices, but early editors were entirely dependent on the definition of signs whose syntax is described in a key mapping. The set of key sequences in advanced editors, such as ⟨Ctrl-X⟩⟨Ctrl-S⟩ (the "save file"-operation in Emacs), are made out of perhaps one or two hundred symbols that typically can be combined in one-, two, or three-symbol sequences. Although most users learn only a fraction of the command vocabulary, all users eventually string together effortlessly a subset of the symbols.

We believe that this ability to sequence simple sequences of actions is deeply rooted, and that mastering such skills is easier than mastering the use of natural language for editing. Indeed, there is evidence that sequential learning is a primate feature, not an exclusively human one[4], whereas natural language appear to be far beyond the abilities of other primates. Even so, non-human primates have been reported to spontaneously sequence pictographs, thereby demonstrating combinatorial processes[5].

The human affinity for sequencing is more impressive: word constituents are stringed together to form words in inflicted languages, alphabetic keys are sequenced as words on the computer, and nodes are stringed together to form music. The evidence seems overwhelming: motivated humans easily acquire and express simpler systems of signs than the whole of natural language. Sometimes systems are expressed manually, sometimes orally, and sometimes through both modalities as in the of case words (enounced or typed) or in the case of music (sung or played).

The specific question of how the mind expresses simple sequences of signs that stand for editing operations was the subject of "The Psychology of Human-Computer Interaction" [6]. In this book, a cognitive model GOMS (Goals, Operators, Methods, Selection rules) based on production rule systems was shown to be successful in predicting the time it takes for users to accomplish editing tasks. A tenet of this approach is: the skill of text editing is acquired through training that turns symbolic problem solving into a set of readily-available methods. These methods are selected by brief cognitive processing when tackling editing situations. The GOMS model thus lends further credibility to our thesis that text editing is mediated through combinatorial processes that do not involve natural language.

1.2. Natural language, efficiency, and mode problem

Our aim is to suggest principled ways for obtaining efficient command and control user interfaces. Having explained the potential and importance of primitive systems of signs, we must briefly touch on the subject of why natural language with its almost infinite richness does not appear to be a solution to the problem, even if it could be understood as well by a computer as by a human.

Paradoxically, a main drawback of natural language for editing is that it is inefficient. When used in its most basic way where just individual nouns are exclaimed, natural language often induce punishments, rather than rewards: inserting "!" three times would become "exclamation mark, exclamation mark, exclamation mark." Saying "insert three exclamation marks" is better, but still slower than using the keyboard.

Human nature is clearly against learning complex systems that do not provide superior performance over already acquired methods. From studies of editing efficiency[6], we know that performance is strongly correlated with the number of keystrokes needed for each editing task. It is reasonable to expect that a similar situation holds for spoken input: the fewer words per task, the better performance.

To give some rough quantitative estimates, we use the keystroke-level model[6, p. 264], where the times for individual operations are estimated as: $\mathbf{K} = .20s$ (pressing a key for an average skilled typist; a modifier key is estimated to also average $\mathbf{K}$); and $\mathbf{H} = .4s$ (home hand(s) from device to device). We regard the arrow keys and the usual six keys above them as separate devices, since they are away from the standard position of the hands. In addition, we have defined $\mathbf{S} = .25s$ to be the average time it takes to pronounce a syllable. With these numbers, the time to type three exclamation marks can be estimated to be $(4 \cdot \mathbf{K} = .8s)$, since the first exclamation mark requires pressing the shift key. The time to say "exclamation mark" three times becomes $3 \cdot 5 \cdot \mathbf{S} = 3.75s$ and the time to say "insert three exclamation marks" becomes $8 \cdot \mathbf{S} = 2.0s$. Thus, for this simple operation, the keyboard is clearly superior by a factor of two or three. Similarly, it might be natural to say "go to the beginning of the line" $(9 \cdot \mathbf{S} = 2.3s)$, but it is not gratifying: the keyboard is three times faster $(\mathbf{H} + \mathbf{K} = .6s)$.

These observations extend to most editing operations that are mimicked by natural language commands. But, there is an additional and severe *mode problem*: in order to distinguish between commands and dictation, the interfaces of current commercial dictation systems require the user to make an explicit pause before and after each command. We estimate that such a user-issued pause will typically have the length $\mathbf{S}$ of one syllable. Indeed, an editing operation to convert a plural "s" into a possessive "'s" would be quick on the keyboard (where we assume that the cursor is at the space after the word): move left two characters, hit apostrophe key, and move right two characters, but rather excruciating by voice: "**pause**, move left two characters, **pause**, apostrophe, **pause**, move right two characters, **pause**", where the extra pauses at the beginning and the end may or may not be necessary. The spoken utterance to fix the "s" above takes $18 \cdot \mathbf{S} = 4.5s$ (neglecting the extremal pauses), whereas the keyboard expression might be estimated to be $3 \cdot \mathbf{H} + 5 * \mathbf{K} = 2.2s$ (neglecting the final homing operation).

The need to insert pauses is a highly unnatural restriction. The reader might ponder the commercial potential of a keyboard with similar restrictions: unless there is a pause between key presses the normal meaning of a command key is suppressed and the label of the key is inserted instead.

To overcome the mode problem, we will already at this point make the decision to make a command language unambiguous by construction: no command language construct should be confusable with common segments of natural language. Consequently, dictation and commands may be fluently interspersed.

1.3. Natural language as it is spoken

The preceding discussion has not touched upon the use of natural language as other than a syntactic disguise for keys. Maybe the potential of natural language is realized only when it is used as it is spoken. With problem domain-specific vocabulary extensions, natural language might become an efficient tool. Even if that was the case, the question is hypothetical: the ability of a machine to understand human language is meager. And, if complex algorithms were invented that would interpret spoken human editing intentions, then the vagueness of human language might still be a serious obstacle to using it efficiently.

1.4. Related Work

That natural language for editing may not be that easy to learn or rewarding to use has been discovered by users. A journalist calls it an "illusion" [7], while she suggests a constructed editing notation that has little to do with natural language. This approach differs from ours in one important aspect: it assumes the **pause** technique for command disambiguation, so there is no emphasis on preventing confusion with natural language. The approach does not seem to identify as many primitive editing concepts and to allow as flexibly sequencing of them as our proposal.

In the area of spoken dialogue systems, the somewhat renegade view that the most value from speech recognition is derived from simplified user interfaces has also been voiced: the *Universal Speech Interface* is characterized by a small, fixed set of key words and ways of using the interface [8]. However, our results probably do not reflect on this approach (or vice versa) since text editing is a much more specialized skill than say inquiring about movie times.

Perhaps the most important problem with dictation systems is poor error correction facilities. Even for transcription tasks [9], the keyboard + mouse combination may be two and half times faster among novice users. Note that efficient editing by voice may help alleviate error correction.

The specialized problem of programming by voice has already received some attention. *VoiceGrip*[10] is a systematic approach to three main problems: (1) the pronunciation of symbols that are not English words, (2) the entering of program constructs, and (3) search. The VoiceGrip technique for (1) is to make symbols pronounceable in natural ways: for example, "CurrRecNum" is pronounced "current record number". This technique might advantageously be combined with ShortTalk. The techniques for (2) and (3) rely on natural language representations of programming constructs, but no general approach to commonplace editing situations that are easily solved by keyboard is suggested.

Contradicting the common hypothesis that natural language by the virtue of being natural is immediately natural to use is the study of Karl, Pettey, and Shneiderman[11]. They showed that when users are asked to use a simple set of commands a natural language such as "page up" instead of using the corresponding keys, task performance may be severely affected for non-trivial editing situations. Although the authors note that mouse activation of commands is slower than speech activation according to their empirical results, they draw the surprising hypothesis that speech itself interferes with thinking: the use of speech for commands effects adversely short-term memory.

We believe that there is a much simpler explanation: the lack of training using the small vocabulary of voice commands is the source of the cognitive load. It becomes natural to use the two words "page up" for the specific effect of moving the page on the screen upwards only after an amount of training that makes their use a trained reflex. Importantly, it may therefore matter less what the specific syntax is, or whether the syntax is "natural" altogether.

2. OVERVIEW OF SHORTTALK

ShortTalk is a collection of symbolized editing concepts that can be stringed together in "phrases", consisting of mostly one or two, sometimes three or four concepts. We discuss some of them; for details see[12].

The forward and backward distinction Editing actions are often expressed relative to the text cursor. Thus direction, forwards or backwards from the cursor, is a primary piece of knowledge implicit in our cognition about editing situations. It would be a waste of mental knowledge not to systematically represent this concept. The ShortTalk solution is simple and terse: the vowel denotes direction. For example, "go aift hello" means place the cursor after the occurrence of "hello" *following* the current position; and "go ooft hello" means place the cursor after the occurrence of "hello" *preceding* the cursor. So, a phonic principle is: "oo" means backward and "ai" means forward.

Actions that may stand alone Pressing the ⟨space bar⟩ is "spooce" for half the syllabic effort of saying "space bar." The same applies to "loon" for ⟨return⟩ usually called "new line" in Natural Language Systems. It saves the user the offense of having dictation such as "the new line is that" misinterpreted (at the expense of an acquatic bird). Keys like up and left arrow have similar mnemonic names: go up becomes "goop", that is "go oop" ("oo" sound for backwards motion alterates the vowel of "up") and up arrow becomes "gloof" for "go left" in a similar manner.

Numbers Scottish "ane" is for one, "twain" for two, "traio" for three, "fairn" for four, and "faif" for five. We do not use higher numbers, since they eye can quickly identify only four or five items. And, counting does not appear to be a productive part of editing. The command "line faif" means "go down five lines". By the ai/oo principle, "line foof" therefore means "go up lines". So, we have ten useful and efficient numerals that eliminate the homonymic confusions among "to", "2", "two", and "too" (among other ambiguities). The numerals are in fact crucial to the disambiguation of commands from dictation, but they can never appear by themselves. That is why "line twain" can be embedded in continuous dictation as a command, while "Mark Twain" still may occur as dictation.

Characters, words, lines, paragraphs... Structural concepts for various kinds of pieces of text are: "char" (as in charcoal) for characters, "word" for words, "line" for lines and "para" for paragraphs. Altogether about 10 such concepts can be combined with numerals, and they form an effective set of tools for just moving the cursor. For example, the command "word twoon" puts the cursor at the second word before the current word.

At this point the number of concepts introduced is little relative to number of sequences: the ai/oo principle, the numerals "ane", "twain", ..., "faif", and ten mostly obvious and known terms for pieces of text yield 100 commands that all correspond to common editing situations.

Common places When a user says "this paragraph" under a natural language paradigm, does he or she refer to the paragraph where the mouse pointer is or where the text cursor is? ShortTalk rejects such ambiguity. Instead, there are mnemonic symbolizations: "hare" is "here" for where the cursor is and "tair" is "there" for where the pointer is. But there are more useful positional concepts not made available in most editors although they are latent in our perception of editing. For example, ShortTalk keeps track of where the cursor was before the last *cursor excursion.* This position called "mairk" marks the end of what was inserted last when the cursor if the cursor is no longer there. Mairk is visually denoted by a brown highlighting of the position. To go to the mairk, the user says "gairk" for "go to mairk." (This concept is borrowed and extended from the GNU Emacs text editor.) Another essential concept is that of the position at the start of the last inserted text. This position is called "loost." Naturally, one goes to "loost", which is marked green, by simply saying "goost."

Actions To capitalize is "caip," to uppercase is "aipper", to fix spacing and capitalization is "fix", etc. So, if after the user said "we helped it going" and the text now looks like "the most we had.we helped it going" with the cursor now being at the end, the utterance "Fix loost" repairs the spacing before "we helped". This operation does not move the cursor. Compare this to reaching for the mouse, moving it to locate the period, then clicking it, then find the keyboard again to delete the wrongly-cased letter, inserting the uppercased one, and inserting spaces, then reaching for the mouse again to reposition the cursor, which can be estimated to take $5.2s$. This example illustrates why the sequencing of elementary concepts makes ShortTalk several times faster than traditional mechanical interfaces in many common editing situations.

Searching The principles are very simple:"baif" is for the position before text to look for and "aift" for the position after. The vowel shift ai/oo determines search direction. In the above example, we might also have said "fix boof we" to fix the problem.

Special characters The ShortTalk name for "!" is "clam" (as in "exCLAMation mark"). So, "clam traio" inserts three exclamation marks. The estimated time $3 \cdot \mathbf{S} = .75s$ compares with typing. All keyboard symbols have monosyllabic expressions. A keyboard symbol cannot occur by itself, but may occur with another symbol such as in "col rye" for ":)". In this way, the speech recognizer can distinguish keyboard symbols and their sequences from natural language.

Summary Altogether, ShortTalk defines approximately 170 general concepts, including 33 names for special symbols ("clam"), 20 window and file actions ("switch", "save"), 12 editing actions ("fix", "caip"), 15 structural designators ("word", "senten"), 10 numerals ("traio"), 10 special keys ("spooce"), 8 text movement commands ("smack'), and 7 search concepts ("boof"). For comparison, a keyboard labels around 150 concepts, but may expose hundreds more.

3. PRINCIPLES OF CONSTRUCTION

The ShortTalk symbolization and syntax illustrate two design principles for spoken command and control interfaces that are integrated with transcription.

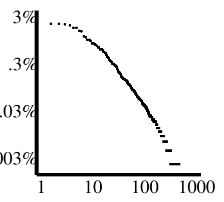
- Identify essential concepts as symbols that receive mnemonic names and that are *stenophonic*, i.e. consisting of preferably only one syllable.

- Construct a highly *orthogonal* (permissive) command grammar. The grammar must be *non-ambiguous*: a parser must be able to identify the commands in each sequence of words that mixes natural language and commands.

Note that one appealingly simple method for achieving non-ambiguity is to make all commands be of the form XY, where only Y is required to not be a word in the natural language (but indeed could be closely related to one, for example by a vowel shift).

4. SHORTTALK: THE CASE STUDY

For approximately 2 1/2 months, the author has recorded all his mouse, keyboard, and speech activity within the Emacs editor, augmented with ShortTalk. The logs, available at [12], reflect e-mail writing, editing technical papers, and programming by an experienced ShortTalk user. Out of 30,000 ShortTalk commands captured, 1060 are distinct. The observed entropy—measured as $\Sigma_{1\leq i\leq 1060} - p_i \log p_i$, where p_i is the observed frequence of the ith operator—is 7.3 bits/operator—and the distribution of operators is pictured below on a doubly logarithmic scale:



The x-axis is the index of a command (the most frequent command has index 1, the second index 2, and so on) and the y-axis is the frequency of the command. The plot seems to illustrate Zipf's Principle of Least Resistance: the frequency of the kth command is approximately $3\%/k$. We measured the average number of syllables per ShortTalk command to be 1.80.

With $\mathbf{S} = .25s$, the use of ShortTalk has averaged 16.1 bits per second of editing information (not counting the arguments of operators). For reference, the entropy of spoken English at a dictation rate of 120 wpm is approximately 16 bps (assuming perplexity of 247 as measured on the Brown Corpus). And, if each ShortTalk concept was bound to a key+modifier combination (of duration $2 \cdot \mathbf{K} = .4s$), then the entropy rate would be $7.3/(1.77 * .4)$bps=10.3 bps, since we have measured an average of 1.77 symbols per command.

The logs capture some 60,000 words of dictation, indicating the importance of spoken commands in practical use: for every two words dictated, a command was issued.

5. REFERENCES

[1] E. Pascarelli and D. Quilter, *Repetitive Strain Injury*, John Wiley and Sons, 1994.

[2] Philip R. Cohen and Sharon L. Oviatt, "The role of voice input for human-machine communication," *Proc. Natl. Acad. Sci. USA*, vol. 92, no. 22, pp. 9921–9927, 1995.

[3] "How May I Help You?$^{\text{SM}}$," http://www.research.att.com/~algor/hmihy/.

[4] Christopher M. Conway and Morten H. Christiansen, "Sequential learning in non-human primates," *Trends in cognitive Sciences*, vol. 5, no. 12, 2001.

[5] E. Sue Savage-Rumbaugh and Duane M. Rumbaugh, "The emergence of language," in *Tools, Language and Cognition in Human Evolution*, Kathleen R. Gibson and Tim Ingold, Eds., pp. 86–108. Cambridge University Press, 1993.

[6] Stuart K. Card, Thomas P. Moran, and Allan Newell, *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Ass., 1983.

[7] Kimberly Patch, "Speech command and control," http://www.scriven.com/RSI/RSIdata/KimsMacros/Kims_Macro_Talk.html, 2002.

[8] Stefanie Shriver, Arthur Toth, Xiaojin Zhu, Alex Rudnicky, and Roni Rosenfeld, "A unified design for human-machine voice interaction," in *CHI*, 2001.

[9] John Karat, Daniel B. Horn, Christine A. Halverson, and Clare-Marie Karat, "Patterns of entry and correction in large vocabulary continuous speech recognition systems," in *CHI 1999*, 1999.

[10] Alain Desilets, "VoiceGrip: a tool for programming-by-voice," *International Journal of Speech Technology*, vol. 4, pp. 103–16, 2001.

[11] L. Karl, M. Pettey, and B. Shneiderman, "Speechactivated versus mouse-activated commands for word processing applications: An empirical evaluation," 1993.

[12] "Shorttalk Web site," http://www.research.att.com/~klarlund/ShortTalk.