1 Introduction

> ...if all my possessions were taken from me with one exception, I would choose to keep the power of communication, for by it I would soon regain all the rest.
>
> Daniel Webster

Extremely slow communication is a daily reality for some people. Rate of expression is significantly and irreversibly reduced by several forms of physical disability, and for those afflicted, unaided spoken and written communication may be intolerably slow—even non-existent. In order for such individuals to participate effectively in society a means of facilitating and accelerating their rate of expressive communication has to be found.

We have developed a device called the *Reactive Keyboard* that accelerates typewritten communication with a computer system by predicting what the user is going to type next. Obviously predictions are not always correct, but they are correct often enough to form the basis of a useful communication device. Since they are created adaptively, based on what the user has already typed in this session or in previous ones, the system conforms to whatever kind of text is being entered. Present implementations have proved most useful in enhancing the command interface to the UNIX operating system by predicting commands, arguments, and filenames; and for the entry of free text.

This paper describes the technical and human interface techniques that are used in the *Reactive Keyboard*. To enable it to make predictions, a model of previously-entered text is created and maintained adaptively. The modeling technique that is adopted was developed for use in text compression, and in fact forms the basis of one of the most effective existing compression methods (originally described by Cleary & Witten, 1984; see Bell *et al.*, in press, for a recent survey of the field). To render it suitable for use in a communication aid, two main issues had to be addressed. First, there were a number of practical problems concerned with resource consumption. Second, whereas for text compression the adaptive prediction mechanism fed an encoder that generated a bit-stream representing the message, for present purposes it must be equipped with a human interface that allows predictions to be displayed and selected. Communication aid users trade physical dexterity for cognitive load, and several difficult interface design issues have to be confronted in an attempt to maximize the accessibility of the predictions and minimize both physical demands and cognitive load. Two different interface styles were selected, one using a plain keyboard and the other a two-dimensional pointing device.

The paper is structured as follows. The remainder of the present section sketches the functional architecture of a communication aid, as a basis for subsequent discussion. The next section introduces the idea of predictive text generation systems, and draws the important distinction between adaptive and non-adaptive models. Section 3 describes the design of the *Reactive Keyboard*, including the prediction mechanism and user interface. Since the success of the device rests very heavily on the success of the adaptive modeling and prediction technique, a discussion is included of some important issues that arise in the implementation of the adaptive predictor. Section 4 gives some practical details about the system, amongst them the size of models, the use of text to prime them, and interaction with local editing facilities. Finally, we conclude with a discussion of the target population and comments from disabled users, and mention some future research directions.

Communication aids can be defined as devices that augment the ability of a communicatively handicapped person to produce communication signals. Several different functional breakdowns of communication aid components have been suggested (e.g. Morasso *et al.*, 1979; Ring, 1980; Rosen & Goodenough-Trepagnier, 1982; Vanderheiden, 1984). Buhr & Holte (1981) identify the four components shown in Figure 1:

(a) *input device* (e.g. joystick, pneumatic switch) which translates the user's physical movements into input signals;

(b) *output devices* (e.g. typewriter, CRT display, speech synthesiser) which display the selected symbol(s)—more than one may be available through a single communication aid;

(c) *selection algorithm* (e.g. linear scan, matrix scan) which uses the input signals to select a symbol from a group of symbols;

(d) *prompting device* (e.g. CRT display, cycling light) which informs the user of the symbols that are currently available for selection and of the actions required to make a selection.

At the core of any communication aid is the "selection set," the group of symbols that are presented to the user for concatenation into messages. The key parameters of the system upon which performance depends are the *number* of elements in the selection set, N; and the number of physical controls or *buttons* in the interface, B (Rosen & Goodenough-Trepagnier, 1982). For example, a conventional keyboard would have $N = B$, were it not for the presence of shift and control keys which double or triple N while increasing B by only one or two.

2 Predictive Text Generation Systems

Predictive text generation (PTG) is a context-sensitive technique for enhancing expressive communication rate. It works by suggesting, on the basis of preceding inputs, what the user wants to select next (Arnott *et al.*, 1984). PTG systems exploit stored information about past selections to predict future ones. Likely continuations are identified by locating a *context* of recent selections in a large "memory" of previously encountered element *sequences*. Frequency-of-occurrence data are then used to sort predictions and offer the most probable continuations to the user for selection, thereby accelerating input.

2.1 The Basis for Prediction

The ability to guess or predict next letters or words relies on the statistical redundancy of language. Shannon (1951) estimated that English is about 75% redundant, and noted that in general, good prediction does not require knowledge of more than a fairly small number of preceding letters of text. While for a native speaker success in predicting English gradually improves with increasing knowledge of the past, apart from some statistical fluctuation, it does not improve substantially beyond knowledge of eight to ten preceding letters (see also Cover & King, 1978; Suen, 1979).

Communication aid designers reasoned that it should be possible to tabulate all length n letter sequences present in a suitably large text sample or word list (e.g. Foulds *et al.*, 1975;

Baletsa *et al.*, 1976; Baletsa, 1977; Heckathorne *et al.*, 1980; Thomas, 1981). The resulting tables might then be used to make predictions of likely next letters or words. Knowing a user's first $n-1$ letter selections would make it possible to predict likely n'th letters by retrieving items that begin with this prefix. The predictions could be preferentially offered to the user in order to speed selection. Alternatively, word predictions could be made by using word prefixes to retrieve likely completions from dictionaries of frequently used words.

Existing PTG systems fall into three main categories: letter anticipators, word or phrase completors and combined systems (Darragh, 1988). Letter anticipators accelerate user input by making likely next letters faster and easier to select. Completors amplify inputs by offering word or phrase completions based on an initial prefix of one or more letters spelled by the user. Word completion strategies form a natural extension to letter anticipation schemes. Combined systems, as the name implies, do both. Most use letter anticipation but switch to word/phrase completion if a known prefix is entered.

2.2 Predictive vs Non-Predictive Systems

Predictive systems differ from their non-predictive counterparts primarily in the way they deal with message composition. Figure 2 illustrates the difference diagrammatically (references to the Figure are emphasized below). In both cases, users *decompose* their *intended message* into a series of *selection elements*. Each component element is then selected using a *sequence of input strokes*. Finally, the system *synthesizes* the series of *selection elements* back into the user's intended *message*. The difference is in the way selection elements are offered to the user. Non-predictive systems present users with a fixed set of selection elements which are equally accessible at each choice point. Predictive systems, on the other hand, use *prediction rules* to make likely elements easier, faster and (possibly) more productive to select.

Figure 2 also illustrates some common PTG system features by expanding the *prediction rules* component. Progressing from right to left, a PTG system monitors user selections (*selection elements*) and stores the last few together in short-term memory to form a *current context*. The context is then sought in long-term memory to find *likely continuation elements* and associated probabilities of occurrence. This information is then processed by the system's *selection algorithm* to generate a *changing selection set display* on the prompting device.

2.3 Adaptive vs Non-Adaptive Models

Every communication aid incorporates a model of the communication task it is designed to facilitate. Models include the language elements the user can choose (the selection set) and the technique for making selections (the selection algorithm). Many aids embody static models, created by the designer and frozen for the lifetime of the device. For example, a predictive system could be primed with a static model of a particular language which captured common words and phrases thought to be representative of a typical user's input. Alternatively, the model could be dynamic, adapting to its user's vocabulary and phrasing.

While systems incorporating static or "canonical" models are general purpose and easy to implement, they lack flexibility and often constrain their users by a largely inefficient language model. The alternative, adaptive, models can either be constructed explicitly by the user ("explicit" modeling) or implicitly by the system ("automatic" modeling). Greenberg

(1984) discusses these alternatives more fully. One of the biggest problems of providing a model explicitly is that users must not only anticipate future language usage but also divert themselves from the task at hand to create the model.

Like explicit modeling, automatic modeling has two distinct stages, model construction and model consultation. Automatic systems construct their models from scratch by continuously monitoring user behavior patterns and updating internal user models to reflect changing behavior. To do so the system must use limited cues and rules to guess likely user behaviors. An automatically constructed model is inevitably probabilistic, and any given prediction may be inaccurate—particularly if user errors are incorporated as though they were part of the user's intended behavior.

2.4 Long-Term Memory

PTG systems form their predictions by consulting long-term memory. This generally contains a large number of recurring selection element sequences (n-grams), with associated occurrence frequencies. The current context, $n-1$ recent selections stored in short-term memory, is used to look up likely continuations. In other words, predictions are made on the basis of the present situation, represented by short-term memory (STM), and past experience, represented by long-term memory (LTM). The prompting device display normally changes after each user selection to present a new subset of predicted elements, in probability order. A four-phase cycle of *user selection,* STM *update,* LTM *lookup, display update* continues for the duration of a message composition session. When adaptive modeling is used, LTM is also updated.

Several types of communication model and memory structure are possible, based on the various levels of redundancy present in natural language. Language redundancies range from orthographic, through to syntactic, semantic, and even pragmatic levels (Pickering & Stevens, 1984). Such models span the disciplines of computational linguistics and artificial intelligence and become increasingly hard to generate and utilize at the higher levels. This paper limits discussion to the n-gram based orthographic models which have been used in all PTG systems to date. These are constructed statically from representative text samples and/or dynamically by monitoring the user's text generating behavior letter by letter. Gathered text experience is stored in LTM as n-gram sequences, with associated occurrence frequencies.

A model's predictive power depends on how accurately its experience matches the user's current communication needs, and on the length (n) and number of stored sequences. Adaptive systems use models that adjust themselves to individual user's idiosyncracies, which is why they are often superior to canonical models. Canonically defined models are giving way to more personalizable explicit and combination models in an effort to gain further communication rate enhancements (Heckathorne & Childress, 1983; Nelson *et al.*, 1984).

2.5 Predictive Text Generation: Pros and Cons

The main advantages of using predictive techniques include communication rate enhancement (Witten *et al.*, 1982; Heckathorne & Childress, 1983); reduced physical load on the user (Baletsa, 1977); reduced cognitive spelling load (Colby *et al.*, 1982; Colby 1984; Gibler & Childress, 1982); and increased articulateness (Heckathorne & Childress, 1983). Rate enhancement is achieved using acceleration and amplification techniques to make the se-

lection of likely language elements faster and more productive. Physical load on the user is reduced coincident with increased overall fluency—less time and effort is expended to generate a message. Spelling errors are reduced by offering most likely letters first, providing a strong visual cue to correct spelling; or by predicting correctly spelled word or phrase completions, forestalling mis-spelling altogether. Finally, the context and frequency sensitivity of PTG systems allows rapid access to very large and rich selection sets. The result is a faster, easier to use, more productive communication aid.

The potential disadvantages of using predictive techniques include unlearnability (Rosen & Goodenough-Trepagnier, 1982); visual vigilance (Rosen & Goodenough-Trepagnier, 1982); and visual discontinuity (Foley & Wallace, 1974). Large memories and constantly changing prompt displays make predictive systems potentially confusing and difficult to learn, and may distract from the primary communication task. This is particularly true when adaptive modeling is used. The visual demand these systems place on their users is a related but more serious concern. Users must constantly direct their attention to the prompting device in order to evaluate offered predictions. Visual workspace allocation is a major difficulty faced by system designers. Workspace must be divided between the prompting device and message composition areas. Selection element menus may consume most of a PTG system's display space, restricting space available for message composition. Visual discontinuity can be partially overcome using multiple windows where one physical device includes both logical displays (Warfield & White, 1983).

Overall, the desirable characteristics seem to outweigh the relatively few disadvantages of predictive techniques for most purposes.

3 The Reactive Keyboard

"To originate is to combine"—Edgar Allen Poe

The idea for the *Reactive Keyboard* evolved from an earlier system called *Predict*, which provided input acceleration for command input to a general-purpose computer system (Witten *et al.*, 1982). It was one of the first communication aids to model user inputs adaptively. Perhaps the most important advance was the use of concatenated predictions. Instead of being single-letter, word, or phrase oriented, the selection set was constructed dynamically from single-character predictions. Predictions were based on an n-gram model which was formed adaptively from the user's inputs.

The *Reactive Keyboard* uses a more sophisticated modeling technique that improves its predictive ability. Principal features of this technique include a tree-structured memory and the use of partial context matching; these are explained further below. Two alternative selection algorithms are provided, tailored for different input devices.

The adaptive model maintains a frequency-ordered list of context-conditioned candidate strings, and presents them to the user either individually or as a menu. It makes each start with a different ASCII character, so that the entire character set can always be accessed. Each option is a concatenation of several predicted characters. With the standard keyboard interface the user can cycle through these predictions and accept all or part of any one. With a proportional input device the user selects not only a menu item but also a point within it, so that part of the item can be chosen.

3.1 The Modeling Technique

A longest-match policy is adopted when seeking likely continuations (in LTM) for the current context (i.e. STM). If no elements in LTM match the current STM in its entirety, the latter is truncated until a match can be found. The technique proves extremely effective because it is capable of predicting from more complete lower-order models if less complete higher-order models fail to contain an instance of the current context. All lower-order models are implicit in the highest-order model so no extra storage is required. This kind of model has been found to be extremely effective for text compression (Darragh *et al.*, 1983; Cleary & Witten, 1984; Bell *et al.*, in press).

The model is stored in a special tree structure that allows partial matches between STM and LTM to be found economically. An example is shown in Figure 3. Here, the phrase "to be or not t" is represented as a tree with maximum depth four. Alternatives at any level are listed alphabetically for presentation purposes only—frequency order is used in practice to speed updating and menu generation. Each node of the tree has four parts: a character element label, a frequency counter, a pointer to the next alternative at its level and a pointer to the next higher level's list of continuations.

A number of important technical issues arise when implementing this partial-match modeling method. Models become quite large and need to be carefully organized because of the real-time search and update requirements imposed by an interactive user interface. Moreover, the system needs to work continuously, with no upper bound on the amount of text that can be accommodated.

Updating LTM

Three operations are required to update and maintain LTM: incrementing frequency counts of recurring n-grams, adding new n-grams to the model as new input behavior is experienced, and removing old n-grams to make way for new ones when the model is full. The update procedure is as follows. Each input character is processed separately. First, if LTM is full, up to n old nodes are deleted to ensure that sufficient space is available for possible additions. (At most n free nodes will be required to store each new input character, one instance at each of n levels.) Then, starting at the highest order, each level in the tree is searched for the input character. If found at any level, the corresponding node's frequency count is incremented. Finally, frequency order is maintained by re-sorting the linked list of alternatives to speed future accesses. If the input character is not found, a new node is created and added to the end of the list with a count of one.

Representing Frequency Counts

It is convenient to represent frequency counts as fixed-length integers, in which case the possibility of overflow must be considered. The *Reactive Keyboard* simply halves frequency counts just before overflow. All associated counts at the current level are also halved. Some accuracy is lost, but the relative frequency of alternatives and the ability to discriminate between them is largely maintained. Any multiplier less than one could be used to reduce counts; one-half works well in practice and is efficiently implemented as a bit-shift operation.

Frequency reduction creates a sort of memory "blurring" even when nodes are not forgotten. The statistical significance of an n-gram slowly deteriorates if it occurs infrequently relative to other alternatives, ensuring that unrepresentative statistics—caused perhaps by

a change in the kind of text being generated—cannot influence prediction efficiency indefinitely.

In order to determine the best size for frequency counts, some text-compression experiments were performed (Darragh *et al.*, 1983). Larger counters will retain probability distributions to greater accuracy, but increase the time-constant for decay. It was encouraging to discover that when the number of bits used for frequency counts was reduced to six or eight for storage reasons, the model actually performed better than earlier versions using thirty-two bit counts. This surprisingly low value indicates that sensitivity to drift in the statistics of the text is more important than accurate representation of the probability distributions. It is quite acceptable to store counts in 8-bit bytes. Seven bits work well in practice and are used in current *Reactive Keyboard* implementations.

Forgetting

Since LTM is finite, exceptional action must be taken when it becomes full. If the system is to continue adapting, less useful n-grams must be discarded to make room for new ones as they occur, and prediction efficiency will depend as much on what the model forgets as on what it remembers.

The *Reactive Keyboard* forgets based on frequency of occurrence, for other automatic schemes would require additional overhead per node stored. The least frequently seen leaf nodes, which are presumably the least likely to recur, are deleted from LTM when new space is required. In addition to keeping LTM within predefined limits, forgetting adds a measure of automatic error correction to the model, for erroneous entries will be the first to be discarded because of their low frequency of occurrence.

When frequency counts are halved as described above, special attention must be paid to nodes whose counts fall to zero. Such nodes could either be retained by restoring their counts to one, or deleted and the space freed for subsequent new nodes. The *Reactive Keyboard* does both by restoring nodes when LTM has room and forgetting them (and their sub-trees) when it is full. Frequency count reduction conveniently doubles as a simple forgetting mechanism.

Data Structures

There are many design options for long-term memory, representing different trade-offs between update speed, retrieval speed, and space consumed. A variety of storage schemes that permit partial matches between STM and LTM to be found have been considered, and some implemented and tested. The method of choice depends on the application context, the total storage space available, and the speed of the implementation on the target computer system.

The aim of the LTM structure is to allow the predictions associated with substrings of STM to be found quickly. A tree-structured storage scheme is essential to avoid repeated searches for each different context length. Trees are represented by including with each node two or more pointers to its children, and three possible storage structures are an n-way tree, a binary tree, and a linked list implementation.

The simplest and fastest, but not the most economical, technique is to include space at each node for a pointer associated with every possible character. For example, using the ASCII character set, 128 pointers are needed at each node—an exorbitant space requirement.

A binary tree representation contains three pointers, and a character, per node. This permits logarithmic-time binary searching for desired characters. Even more compact, but potentially rather slower, is to store a linked list at each node which records the characters associated with it, as illustrated in Figure 3. This requires just two pointers, but involves a time-consuming linear scan through the list for the desired character. This penalty can be reduced by maintaining the linked lists in frequency order. Further details of these options are given by Darragh (1988).

3.2 User Interface

The *Reactive Keyboard* deliberately maintains a clear separation between modeling aspects and user interface aspects. In practice, it is necessary to restrict the range of possible user interfaces to a small number of workable options which can be implemented and (ideally) evaluated against one another. The most suitable interface can then be selected for a given user's circumstances.

Input Devices and Implementations

The user interface is determined by the user's input device and is adaptable to a wide variety of two-state and proportional devices. Two different interface options have been developed, one keyboard (or button) based and the other mouse (or pointer) based. The former was designed for remote access to a host computer via a telephone link and uses a standard VDU keyboard and screen for input and output. In this version, called *RK-button*, predictions are presented at the host cursor one at a time, and users can step through and select any one of the possible continuations offered by the predictor. The latter, called *RK-pointer* and illustrated in Figure 4, uses a proportional input device for menu selection, and incorporates a dual window display which separates the menu from the text being entered. Both employ the same two-dimensional menu structure, but they display and access it quite differently.

Two-Dimensional Menu Generation

The *Reactive Keyboard* generates the initial characters of its predictions using a simple ordering strategy that favors longer contexts. First, any matches found for the highest-order context are added to the menu in frequency order. Then, progressively shorter context matches are made and any new character predictions which have not yet been encountered are added to the list, again in frequency order. This ensures that each initial menu character is unique and limits the total number of items to the size of the ASCII alphabet. When the context length becomes zero, all remaining characters not yet found while context matching, but which have occurred in the input stream, are added in frequency order. Finally, all remaining characters which are not represented in the model are added to the list.

Figure 5 gives an example of how an initial element list would be generated if the user were spelling the word "HAPPY" and had already entered "HAPP". It is based on a 26 letter alphabet plus space, with $n = 4$. The alternative predictions for each context length are shown in frequency order. Notice that the relative popularity of candidate letters varies with order, and that once a letter appears at a given level it reappears at all lower levels too. Letters not previously encountered at higher levels are added to the menu list in frequency order—each level's new additions are underlined in the Figure.

Once the list of initial letters is complete, each one in turn is followed "into the future" to complete the menu item. Progressively longer concatenations are made by assuming that the current prediction is correct and destined to be accepted. In the present example, the $n-1$ context "APP" would be shifted forward to make concatenated predictions. The process repeats with the projected new context until either the end-of-line character is predicted or the screen line length is exceeded. For example, if the current context "APP" predicted the initial letter "Y", then "Y" would be assumed correct and the context shifted to "PPY". This would then be used to predict another letter, say "␣", and the process repeated with the projected context "PY␣." If no further elements were predicted, the concatenation "Y␣" would be displayed for selection.

Menu Selection

RK-button has five function keys associated with item selection: "previous" and "next" keys to step through options, and three "accept" keys which accept characters, words, and complete lines respectively. With *RK-pointer*, where the user points to a position within a menu item, the substring up to that point—referred to as the "extent"—is inserted into the text buffer. For example, if the prompting display included a menu item "next prediction," then pointing to the letter "d" would insert the extent "next pred".

Immediately after a menu selection is made, a new set of menu items is generated and displayed. This involves first updating STM to take the selection into account, and then consulting LTM to generate a new set of concatenated predictions. In *RK-pointer*, LTM is updated as soon as the selection is made. In *RK-button*, line buffering is used and LTM updated only after the newline character is selected, because the system provides line-editing facilities that the user frequently employs while composing each line.

The selection feedback provided by *RK-button* simply involves moving the text cursor past the selected item (character, word, or line) and generating a new prediction. *RK-pointer* gives selection feedback in two locations on the display: in the text buffer, by transferring the currently highlighted menu item/extent to the cursor position, and in the menu window (Figure 4). Feedback in the menu window is more complex, though it changes in a predictable way after every selection. The mouse cursor remains at the current menu position, and most of the selected item/extent is shifted out of the menu window, leaving a few characters as additional context to guide further selections. This aids item selection because the beginning letter or letters of a word, and its overall length, are primary visual cues for word discrimination (Dunn-Rankin, 1978; Nooteboom, 1981).

Displaying the Menu

RK-pointer's prompt display consists of a menu window separate from the host-controlled text buffer. The user may position and size the menu window using standard window system commands. The best position relative to the text buffer depends on both the display device's physical dimensions and the user's personal preferences. By varying its size the user has direct control over the number and maximum length of predictions in the menu. The menu never exceeds 128 items because the first letter of each item is unique, and in practice is usually about 90.

What is the best menu length? Human factors considerations would limit the number of items in a window to something like Miller's "magic number" seven plus or minus two

(Miller, 1956). When the system is primed by sufficiently representative text, it is possible to achieve high hit rates with menus of this size. For example, simulation studies have been reported for various levels of priming (Witten *et al.*, 1983). By plotting the probability of selection against menu length, these demonstrated (for a specific sample of text) that a window with ten items would contain 99.8% of all desired initial characters if LTM were pre-primed with the same text sample. Even without any priming, on a fairly short 11,000 character passage the first ten items contained 69% of the desired initial letters. These results suggest that given adequate priming, menus of about six items or more would suffice to ensure that appropriate selections could virtually always be made without having to scroll or page further down the menu. In practice, maximum menu length is left to the individual user's preference.

Truncating Menu Items

The maximum size of individual menu items is limited only by the width of the prompting device. There are several possible strategies for determining it. For example, one can take a fixed-length approach and fill all available menu space. Alternatively, length could be determined as a function of a prediction's popularity and/or the context length associated with retrieval of its initial character. A further possibility is to base the length on perceptual units such as words or lines of text.

The first method is easily implemented and is used in current versions of the *Reactive Keyboard*. Besides its simplicity, this technique offers the user the greatest potential selection productivity. The second method has some merit in that item length would provide feedback on the relative strength of competing predictions, but this does not add appreciable information when items are displayed in popularity order. It is supported by evidence that the grammatical correctness of n-gram based predictions increases with model order and decreases with prediction length (Damerau, 1971).

The third method, unlike the first two, takes account of the grammatical structure of predictions and the significance of delimiting white space. The most general approach is to create fragments of some predetermined maximum length, and then shorten them to suit the particular psycho-linguistic, display and selection strategies in effect. One drawback is that large variations in item length may occur due to the relative positions of various delimiters.

A minimum item length of approximately seven characters may be desirable based on the observation by Salthouse (1984) that a preview size of seven letters is required for efficient copy typing by normal subjects. Another approach to determining the minimum item length is to look at mean *type* (unique word) and *token* lengths, the latter being a frequency-weighted version of the former. Taking words (or types) into account suggests that a minimum length of about twelve characters is required to ensure that 85%–90% of words can be presented unbroken in the menu (Kucera & Francis, 1967). About the same number of characters could be retained as context feedback for word recognition in the menu buffer. Longer items help decrease typographical errors, reducing overall error recovery time.

Ordering Menu Items

Once the number and length of menu items are determined, they must be sorted into a sensible order for display on the prompting device. Clearly each successive menu page should contain items of decreasing popularity. There is, however, considerable latitude for more elaborate ordering schemes within individual menu pages. Current implementations simply present predictions in probability order. Alternatives are plain alphabetical ordering or cursor-centered placement, where high-probability items are positioned near the cursor. Furthermore, it might be desirable to minimize changes between one display and the next by retaining the position of items whose initial characters also appeared on the previous menu.

The final display issue to be considered is the representation of non-printing characters (other than the space character). ASCII character-oriented prompting devices limit the possibilities. We adopt the common convention of displaying control characters as a digram, prefixing the uppercase rendition of the character with a "^". For example, the BEL and NL characters are displayed as "^G" and "^J" respectively. The most obvious disadvantage of this technique is that it impinges on available display space. In addition, such digrams can be awkward to recognize, particularly when several are presented in close proximity. The number of control codes displayed is dramatically reduced when line buffering is used, as in *RK-button*, because no attempt is being made to model the intra-line editing operations (backspace, line erase, etc).

4 Integrating the Reactive Keyboard into a System

Our description so far has concentrated on the new communication medium that the *Reactive Keyboard* provides: adaptively predicted text, offered to the user for selection. When this new medium is embedded within a system, a number of important practical considerations arise that greatly affect its useability. The success of the modeling technique hinges on the parameters of the model—its maximum order n and the amount of space allocated to it—and on the text used to prime it. It is often worthwhile to explicitly alter the system's model of the user's communication requirements by re-priming it with new samples of text. User control of display parameters, such as menu size, has already been mentioned. An important issue is how the predictions interact with line-editing facilities such as the character, word, and line erase functions. Another is the possibility of segmenting the input stream based on knowledge of what the dialogue means. Finally, there is the question of getting help on use of the system itself.

4.1 Model Size

The maximum order n of the model—which is governed by the size of STM—determines the depth of the tree that is represented in LTM. Experiments in text compression show that the predictive power of the model is relatively insensitive to the value of n provided it exceeds a certain minimum (Cleary & Witten, 1984), and current implementations of the *Reactive Keyboard* use n=7.

The second parameter of the model is its size, in other words the total number of n-grams it accommodates. It is difficult to estimate accurately the minimum size required to store a complete model of an open-ended stream of text. In practice, limits are imposed on

the number of nodes by implementation considerations such as the amount of storage space available, the size of each tree node and the type of data structure employed. Versions of the *Reactive Keyboard* have been implemented with both the two-pointer linked list structure and the three-pointer binary tree structure mentioned in Section 3.1. *RK-button*, intended to accelerate command input to a computer system, uses the former, and 64,000 nodes appears to be sufficient for its constrained command input predictions; these occupy 640 Kbytes. *RK-pointer*, intended for free text entry as well as command input, uses the latter and can accommodate up to 16,000,000 nodes, which would occupy a total of around 100 Mbytes (see Darragh, 1988).

4.2 Priming the Model

By far the most important factor affecting the quality of predictions is the actual text used to prime the model. Finding truly representative text samples is difficult, as the statistical characteristics of seemingly similar samples can vary greatly (Kucera & Francis, 1967; Gibler, 1981). This implies that the text samples used to prime LTM must be carefully selected to be as representative as possible of what the user wants to generate.

Sources of Text for Priming

The *Reactive Keyboard* derives its model in three ways using a combination of automatic and explicit modeling techniques. Priming occurs (a) automatically from a default (or user-specified) startup file, (b) automatically from current user inputs, and (c) explicitly from any text file the user chooses to add into the model. Automatic modeling keeps LTM up to date, yet users have the option of explicitly specifying (and even, if they wish, altering) the text used to prime it. In the current UNIX implementation the default startup text file is simply a log of all text generated to date by the user with the *Reactive Keyboard*. After priming LTM with the startup file, the initial model and log file are updated automatically as new text is generated. More recent entries always take precedence when LTM is full, through "forgetting" mechanisms discussed earlier.

Global context control is available during a session because the user can at any time request a text file to be read by the *Reactive Keyboard*. This feature also allows LTM to serve as a store in which the user may capture program output (such a list of filenames) on a temporary basis, for possible future use in text generation. Moreover, LTM can be erased and re-primed, giving the user the option of either keeping multiple contexts concurrently in the model, or—by first clearing LTM—switching between separate, possibly very different, global contexts.

Establishing the Character Set

While the entire ASCII character set can be used as n-gram elements, many of these never occur in normal text generation. Analysis of several fairly large ASCII text files—both English and program source code—revealed that about 30 to 40 characters were not represented at all. Moreover, whenever LTM starts unprimed, even common symbols initially have no frequency listing in the model. For example, only the letters "#,b,e,n,o,r,t" are contained in the unprimed model in Figure 3; all other symbols have zero frequency so far.

To establish initial character frequencies, the *Reactive Keyboard* reads a standard (or user-defined) frequency table before starting a session. This contains a frequency-ordered

list of the symbol set derived from a large sample of representative text. It is used to build a backup zero-length context level into the model which is scanned after the adaptive part of LTM. Symbols with equal frequency are listed in the table in collating order, and ones that do not occur in the source sample are therefore presented last in ASCII sequence.

Editing Priming Files

It is rarely necessary to edit the *Reactive Keyboard's* priming file to eliminate errors, since they become low-probability predictions that are eventually discarded by the forgetting mechanism. There is, however, a practical reason for editing it that arises primarily from exclusive reliance on a frequency-based prediction heuristic. At times, before log file has grown sufficiently large, there may be a desire to override certain high-frequency items explicitly in favor of more recent, yet lower-frequency, items. The most common example in our experience occurs when an electronic mail correspondent changes addresses. The *Reactive Keyboard* will faithfully predict the old address instead of the new one, until the latter has been seen more often than it. To circumvent this problem one can edit the log file and either delete occurrences of the old address or, better still, change them all to the new address to preserve the historical context. Of course, such action is not really necessary since the new address will be predicted as an alternative to the old and will eventually automatically preempt it. Editing the address simply forces the change and speeds future correspondence by making the correct address the first to be offered.

4.3 Interaction with Local Editing

An important choice in predictive text generation is the level at which user behavior is modeled. Specifically, should user behavior be modeled in its entirety, including input line editing and control functions, or should some or all of these user interface tasks be isolated so that only finished text is modeled? Whiteside *et al.* (1982) studied able-bodied knowledge workers creating documents, and secretaries transcribing and updating documents. They attempted to record and describe actual usage patterns by keystroke-level logging of free use of different text editors. They discovered that only about one-half of users' keystrokes were for text entry, another quarter were for cursor movement, an eighth were for deletion and the rest were for miscellaneous functions. Their results suggest that it may not be sufficient just to model the final form of the text generated if maximal predictive assistance is to be provided.

Previous PTG systems have based their canonical communication models on representative samples of completed text rather than on the user's actual behavior. The alternative is to model interactive user text generation behavior and not simply the finished product. However, as Nelson *et al.* (1984) point out, an aid which learns its user's *mistakes* will not be very helpful. A simple strategy to minimize the assimilation of errors is to delay updating the model until each input line is complete. Specific line editing capabilities (which are not modeled) must then be incorporated into the PTG system's user interface.

RK-pointer and *RK-button* take different approaches to this issue. *RK-pointer* models all user behavior, including cursor movement and deletion functions, in an effort to provide greater text generation assistance. *RK-button*, on the other hand, isolates line-buffer editing functions at its user interface by supplying explicit line editing commands; and only the finished text is modeled.

Both approaches have merits and drawbacks. Modeling all behavior gives added assistance in non-text entry tasks, but interspersing less redundant or predictable edit and control operators among what would otherwise be connected text effectively lowers n for the text. Prediction and display of non-text operators can also be problematic and may unduly clutter the prompt display. The main advantage of modeling only finished text is that many potential errors are filtered out at the user interface. The best approach for a specific user depends on the application and the user's individual ability and circumstances.

4.4 Segmenting the Dialogue

Just as it is possible to exclude certain user behavior (like line editing) from the model, special elements can be added which do not normally occur in the input stream, but which nevertheless serve to guide predictions.

An example is the special character which the *Reactive Keyboard* inserts to mark the beginning or end of each text generation session. This marker is added to synchronize STM with LTM at the start of a new session before any new user generated context is available. This alignment is based on the reasonable assumption that users will generate predictable command sequences when logging in (such as reading new mail).

It would be very easy for *RK-button* to monitor the commands issued by the user, tracking the state of the system, and alter its operation accordingly. For example, it could re-prime LTM when an editor is entered, and keep different priming files for different editors (mail, general text, programming languages) or for when the same editor is invoked on different file types. However, we have resisted the temptation to provide this kind of integration, partly to keep the system general-purpose, partly because there is often a surprising amount of cross-over between the contents of different file types (e.g. natural-language comments in a program file; filenames in mail), and partly because we want to rely on a single mechanism of adaptation to track changes of context.

4.5 Getting Help

The final meta-level task is getting help on the working of the *Reactive Keyboard* itself. Because of its interface transparency, *RK-pointer* has no on-line help facility. Though this may seem counter to human factors principles, it is noteworthy that the "help" key available to subjects in a pilot experiment with the older *Predict* system was *never* used. Apparently this type of PTG system is easily understood and used after receiving brief introductory instructions. *RK-button*, on the other hand, includes a help facility listing available commands and their functions—most of which are related to line-buffer-editing meta-tasks (e.g. delete word).

5 Discussion and Conclusions

How much assistance does the *Reactive Keyboard* actually provide? The answer depends on the user's range of residual communication abilities and the environment they are working in. Evaluating adaptive PTG systems is extremely difficult and formal user interface experiments have not yet been conducted. Further studies could perhaps determine such things as optimal menu display and selection feedback strategies. However, these are likely to vary

greatly with individual users and the nature of their disability, and this reduces the utility of formal evaluation.

For example, a good typist may find the *Reactive Keyboard* a hindrance in any environment where a full keyboard is available. However, it might prove invaluable as an occasional text generation facility in an (otherwise) keyboard-less graphics interface. A moderate to poor typist—or a physically limited one—may find *RK-button* worthwhile for mundane command input, but too visually distracting or awkward for general text generation. A person with no keyboard experience such as a young child, or someone who cannot use a standard keyboard efficiently such as a high-level quadraplegic, might find *RK-pointer* indispensable as a writing aid.

Of all potential users, those with severe physical limitations and communication disabilities stand to gain the most from the *Reactive Keyboard*. Certain individuals within this user group will find it a valuable time- and energy-saving enhancement to (or replacement for) their standard communication aid when writing or accessing computer systems.

5.1 Comments From Two Disabled Users

Two physically disabled students at the University of Calgary currently use *RK-button* as their standard command interface. Their combined daily experience amounts to approximately three years. One user types with one partially paralyzed hand and uses the system for entering commands and electronic mail. He estimates that over a two-year period, it has provided assistance on over thirty thousand host system commands, averaging 10 predicted characters/command, and writes:

> The *Reactive Keyboard* has dramatically changed the way I use computers. I now use much longer, more descriptive, file names than I otherwise would have without its reliable recall and typing assistance. I also rely completely on *RK-button* to remember such things as electronic mail addresses and long complex command-line sequences. Life on-line would just not be the same without it.

The other user has a progressive neuro-muscular disorder and is extremely slow and otherwise unable to write. He uses *RK-button* as a general command interface and writing aid.

> I find the *Reactive Keyboard* to be an extremely beneficial tool for typing. Since I have severe neurological damage in my hands, it seems to cut the time I spend coding manyfold. To illustrate this, I need only inform you how I mailed John this letter. All that was required was my typing "ma", after which time it predicted "il darragh^J". So, I was able to type and enter a command normally requiring thirteen keystrokes in but three! This saves much time since it is rather difficult for me to access some keys on the board. It must be remembered that both my hands and fingers move slowly and inaccurately.

5.2 Ideal Technology

The *Reactive Keyboard* is adaptable to a wide range of technologies. The ideal is an implementation resident in the user's personal workstation from which the local operating

system and other host computers can be accessed. Fully transparent to the local workstation, it would perhaps constitute a plug-in card complete with LTM, microprocessor, and prediction display software. Though *RK-button* would not involve any special display-space considerations, *RK-pointer* would have to integrate into the workstation's window system to construct and update its prompting display. Certain cases might require a separate prompting display to eliminate display space conflicts. For example, a 8×25 character flat panel display, plugged into the *Reactive Keyboard's* add-on card, could be positioned near the standard display. The precise configuration would have to depend on the user's input device and a number of other implementation-specific factors.

5.3 Summary

The *Reactive Keyboard* combines and exploits many PTG concepts and represents a considerable advance over earlier systems. Table 1 summarizes its major advantages and shortcomings. The attention given to both the predictive modeling technique and user interface issues results in a synergy that is unprecedented in communication aid design. The *Reactive Keyboard* seems to have great potential to enhance the ease and rate of communication for physically limited people.

Acknowledgements

This research has been supported by the Natural Sciences and Engineering Research Council of Canada and by the Alberta Heritage Foundation for Medical Research. We would like to acknowledge the enthusiastic support of David Hill throughout the work.

References

Arnott, J.L., Pickering, J.A., Swiffin, A.L., and Battison, M. (1984) "An adaptive and predictive communication aid for the disabled exploits the redundancy in natural language" *Proc 2nd International Conference on Rehabilitation Engineering,* 349-350, June.

Baletsa, G.S., Foulds, R.A., and Crochetiere, W. (1976) "Design parameters of an intelligent communication device" *29th Annual Conference on Engineering in Medicine and Biology,* 371, Boston, MA, November.

Baletsa, G.S. (1977) "Anticipatory communication" Masters Thesis, Engineering, Tufts New England Medical Center, Medford, MA, July.

Bell, T.C., Cleary, J.G., and Witten, I.H. (in press) *Text compression.* Prentice Hall, Englewood Cliffs, NJ.

Buhr, P. and Holte, R. (1981) "Some considerations in the design of communication aids for the severely physically disabled" *Medical and Biological Engineering and Computing, 19,* 725-733.

Cleary, J.G. and Witten, I.H. (1984) "Data compression using adaptive coding and partial string matching" *IEEE Trans Communications, COM-32* (4) 396-402, April.

Colby, K.M., Christinaz, D.U., Parkison, R.C., and Tiedemann, M. (1982) "Predicting word-expressions to increase output rates of speech prostheses used in communication disorders" *Proc IEEE International Conference on Acoustics, Speech and Signal Processing,* 751-754.

Colby, K.M. (1984) "Intelligent speech and memory prostheses" *ACM SIGCAPH Newsletter* (34), Spring.

Cover, T.M. and King, R.C. (1978) "A convergent gambling estimate of the entropy of English" *IEEE Trans Information Theory, IT-24* (4) 413-421, July.

Damerau, F.J. (1971) "Markov models and linguistic theory: an experimental study of a model of English" *Janua Linguarum* (95).

Darragh, J.J., Witten, I.H., and Cleary, J.G. (1983) "Adaptive text compression to enhance a modem" Research Report 83/132/21, Computer Science Department, University of Calgary.

Darragh, J.J. (1988) "Adaptive predictive text generation and the Reactive Keyboard" MSc Thesis, Department of Computer Science, University of Calgary.

Dunn-Rankin (1978) "The visual characteristics of words" *Scientific American, 238* (1) 122-130, January.

Foley, J.D. and Wallace, V.L. (1974) "The art of natural graphic man-machine communication" *Proc Institute of Electrical and Electronic Engineers, 62* (4) 462-471, April.

Foulds, R.A., Baletsa, B.S., and Crochetiere, W.J. (1975) "The effectiveness of language redundancy in non-verbal communication" *Proc Conference on Devices and Systems for the Disabled,* 82-86.

Gibler, C.D. (1981) "Linguistic and human performance considerations in the design of an anticipatory communication aid" PhD Thesis, Northwestern University, Evanston, Illinois, June.

Gibler, C.D. and Childress, D.S. (1982) "Language anticipation with a computer based scanning aid" *Proc IEEE Computer Society Workshop on Computing to Aid the Handicapped,* 11-15, Charlottesville, Virginia, November.

Greenberg, S. (1984) "User modeling in interactive computer systems" MSc Thesis, Computer Science Department, University of Calgary.

Heckathorne, C.W., Doubler, J.A., and Childress, D.S. (1980) "Experience with microprocessor-based aids for disabled people" *Proc IEEE Computer Society Workshop on Applications of Personal Computing to Aid the Handicapped,* 53-56, April.

Heckathorne, C.W. and Childress, D.S. (1983) "Applying anticipatory text selection in a writing aid for people with severe motor impairment" *IEEE Micro,* 17-23, June.

Kucera, H. and Francis, W.N. (1967) *Computational analysis of present-day American English.* Brown University Press, Providence Rhode Island.

Miller, G.A. (1956) "The magical number seven, plus or minus two: some limits on our capacity for processing information" *Psychological Review, 63* (2) 81-97, March.

Morasso, P., Pesno, M., Suetta, G.P., and Tagliasco, V. (1979) "Towards standardization of communication and control systems of motor impaired people" *Medical and Biological Engineering, 17,* 481-488.

Nelson, P.J., Korba, L.W., and Park, G.C. (1984) "Evolution of the MOD keyboard system" *Proc IEEE Computer Society 3rd Annual Workshop on Computing to Aid the Handicapped,* 3-10.

Nooteboom, S.G. (1981) "Lexical retrieval from fragments of spoken words: beginnings vs endings" *Journal of Phonetics, 9,* 407-424.

Pickering, J.A. and Stevens, G.C. (1984) "The physically handicapped and work related computing: towards interface intelligence" *Proc 2nd International Conference on Rehabilitation Engineering,* 126-127, June.

Ring, N.D. (1980) "Communication aids for the speech impaired" in *The use of technology in the care of elderly and the disabled: tools for living,* edited by Bray, J. and Wright, S., pp 79-82. Greenwood Press, Westport, Connecticut.

Rosen, M.J. and Goodenough-Trepagnier, C. (1982) "Communication systems for the nonvocal handicapped: practice and prospects" *Engineering in Medicine and Biology Magazine,* 31-35, December.

Salthouse, T.A. (1984) "The skill of typing" *Scientific American, 250* (2) 128-135, February.

Shannon, C.E. (1951) "Prediction and entropy of printed English" *Bell System Technical J,* 50-64, January.

Suen, C.Y. (1979) "n-gram statistics for natural language understanding and text processing" *IEEE Trans Pattern Analysis and Machine Intelligence, PAMI-1* (2) 164-172, April.

Thomas, A. (1981) "Communication devices for the nonvocal disabled" *IEEE Computer,* 25-30, January.

Vanderheiden, G.C. (1984) "The spectrum of communication technology needs of individuals with communication impairments" *In extension course handout, Personal Computers for the Handicapped, University of Alberta,* May, adapted from "Non-conversational communication technology needs of individuals with

physical handicaps'', *Rehabilitation World 7* (2), Summer 1983.

Warfield, R.W. and White, G.M. (1983) ''The new interface technology'' *Byte, 8* (12) 218-230, December.

Whiteside, J., Archer, N., Wixon, D., and Good, M. (1982) ''How do people really use text editors?'' *Proc ACM SIGOA Conference on Office Information Systems,* 29-40, June 21-23.

Witten, I.H. (1982) ''An interactive computer terminal interface which predicts user entries'' *Proc IEE Conference on Man-machine Interaction,* 1-5, Manchester, England, July.

Witten, I.H., Cleary, J.G., and Darragh, J.J. (1983) ''The reactive keyboard: a new technology for text entry'' *Converging Technologies: Proc Canadian Information Processing Society Conference,* 151-156, Ottawa, ON, May.

List of Tables

List of Figures

Advantages

- ⊕ Simple user interface is easy to learn; one operation mode.
- ⊕ Adaptable to diverse users, computers and environments.
- ⊕ Reduces typing errors and recovery time.
- ⊕ Maximizes motor ability of users with low activity tolerance.
- ⊕ Speeds users whose input rate is dominated by movement time.
- ⊕ Complete transparency to host system; runs on minis/micros.
- ⊕ Works in harmony with OS level amplification/abbreviations.
- ⊕ *RK's* keyboard emulation is adaptable to most input devices.
- ⊕ Concatenated predictions are based on adaptive frequencies.
- ⊕ Combination PTG predicts chars/words/fragments/phrases.
- ⊕ Exploits sub-word fragments and trans-word redundancies.
- ⊕ The selection set presentation not bound to LTM's token set.
- ⊕ Works with natural/artificial language; alphabet independent.
- ⊕ Automatic modeling; no explicit language usage evaluation.
- ⊕ Prediction technique uses lower n if higher n model not yet formed.
- ⊕ Priming can be chosen according to text type to be entered.
- ⊕ No limit on priming source type or size; context specific.
- ⊕ Proven LTM prediction efficiency; text compression results.
- ⊕ Multiple LTM global contexts; separate or simultaneous LTM.
- ⊕ Amplification reduces *Cost* (sel/word); possibility of $Cost < 1$.
- ⊕ Acceleration reduces *Length* (stroke/sel) speeding input.

Disadvantages

- ⊖ Constant visual demand and dependency; visual discontinuity.
- ⊖ Display space is commandeered; control code display awkward.
- ⊖ Additional metadialogue required to control user interface.
- ⊖ Constant change; large dynamic LTM is basically unlearnable.
- ⊖ Can generate predictions that were not in the input stream.
- ⊖ Recency information is discarded; time is largely ignored.

Table 1: Reactive Keyboard—advantages and disadvantages

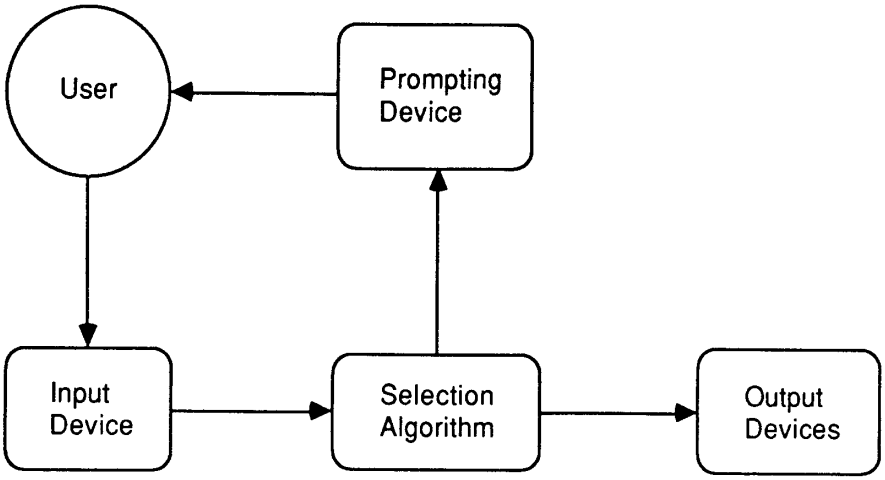

User
Prompting Device
Input Device
Selection Algorithm
Output Devices

Figure 1

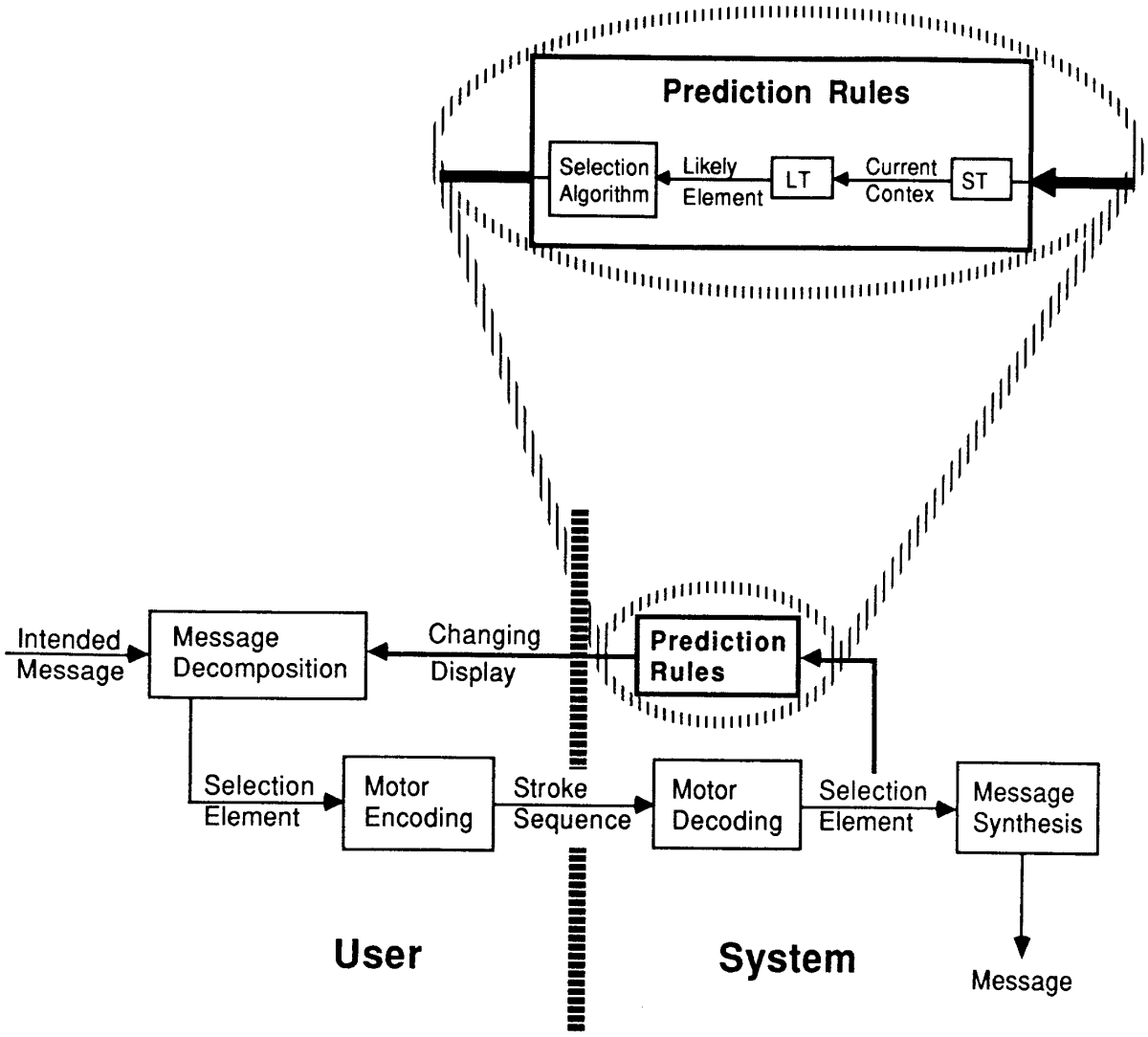

Prediction Rules
Selection Algorithm
Likely Element
LT
Current Contex
ST
Intended Message
Message Decomposition
Changing Display
Prediction Rules
Selection Element
Motor Encoding
Stroke Sequence
Motor Decoding
Selection Element
Message Synthesis
User
System
Message

Figure 2

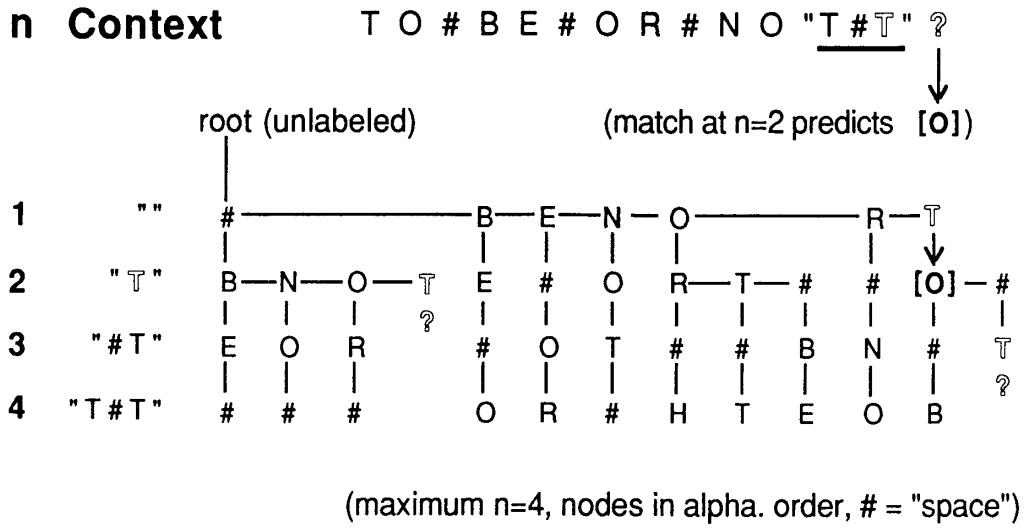

n Context
T O # B E # O R # N O "T#T" ?
root (unlabeled)
(match at n=2 predicts [O])
1 ""
2 "T"
3 "#T"
4 "T#T"
(maximum n=4, nodes in alpha. order, # = "space")

Figure 3

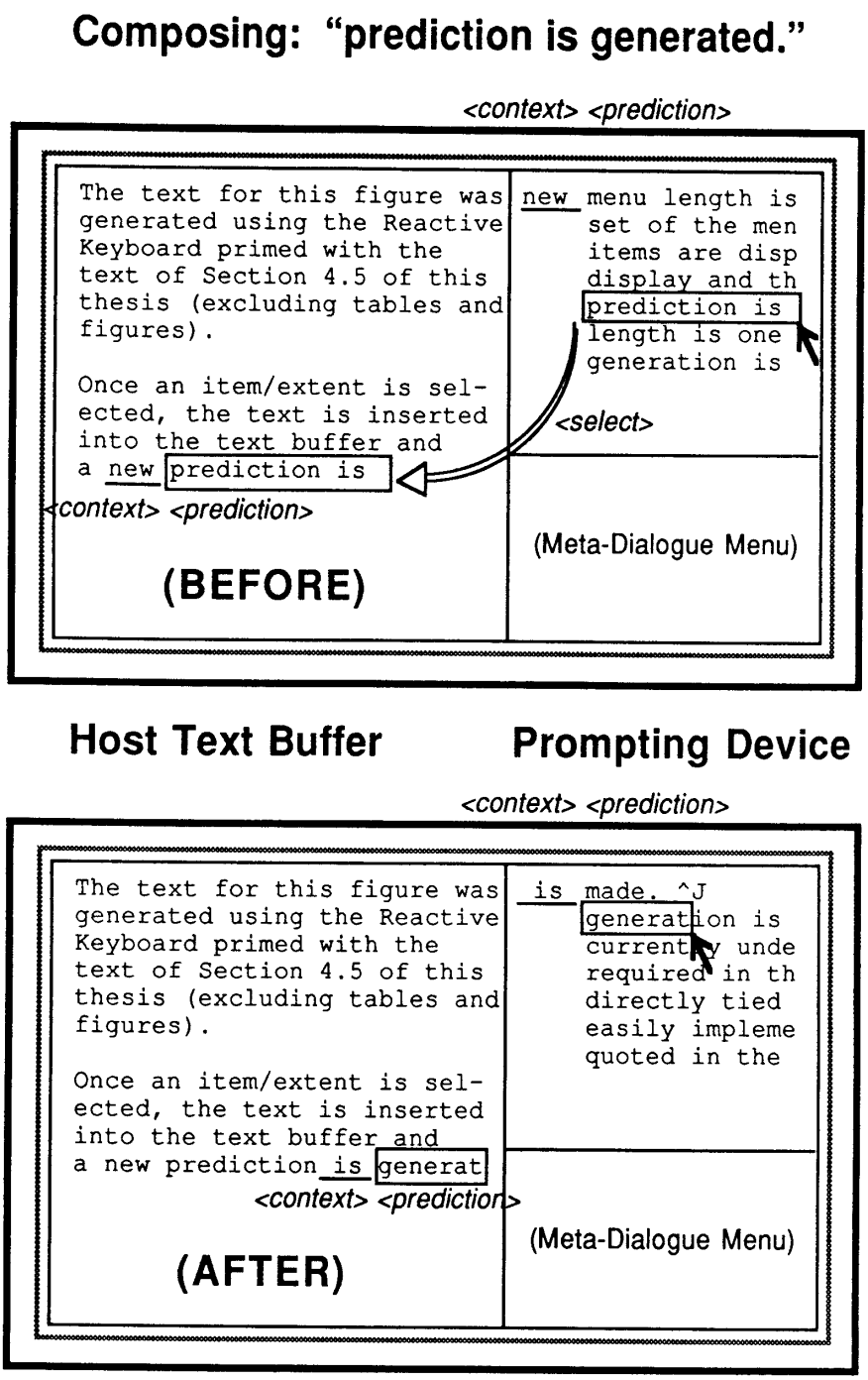
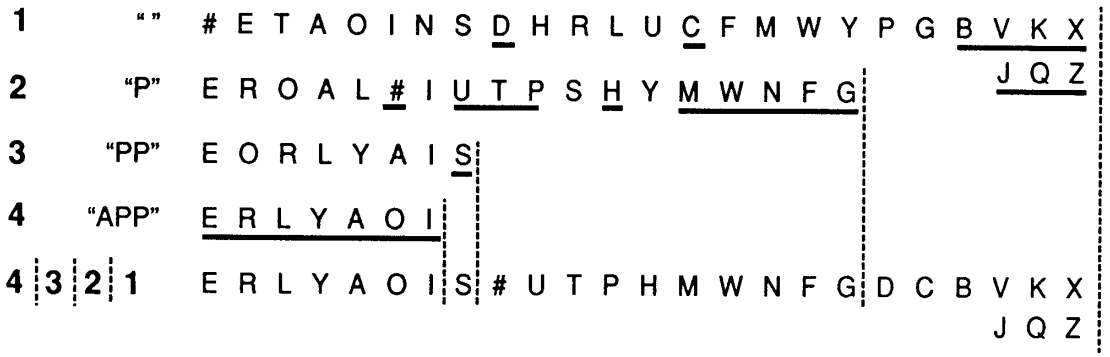

Composing: "prediction is generated."
<context> <prediction>
The text for this figure was generated using the Reactive Keyboard primed with the text of Section 4.5 of this thesis (excluding tables and figures).
Once an item/extent is selected, the text is inserted into the text buffer and a new prediction is
<context> <prediction>
(BEFORE)
new menu length is set of the men items are disp display and th prediction is length is one generation is
<select>
(Meta-Dialogue Menu)
Host Text Buffer
Prompting Device
<context> <prediction>
The text for this figure was generated using the Reactive Keyboard primed with the text of Section 4.5 of this thesis (excluding tables and figures).
Once an item/extent is selected, the text is inserted into the text buffer and a new prediction is generat
<context> <prediction>
(AFTER)
is made. ^J generation is currently unde required in th directly tied easily impleme quoted in the
(Meta-Dialogue Menu)

Figure 4



(maximum n=4, letters in freq. order, # = "space")

Figure 5