SUITEDasher – A Multilingual Keyboard and Mouse Interface for Motor-Impaired Users

David Lyle and Bill Manaris

Computer Science Department, College of Charleston
66 George Street, Charleston, SC 29424, USA
dlyle@edisto.cofc.edu, manaris@cs.cofc.edu

Abstract

This paper presents the design of SUITEDasher – an open-source, multilingual speech user interface for motor-impaired users. SUITEDasher's architecture is based on SUITEKeys – a speech user interface for manipulating a speech keyboard and mouse. SUITEDasher has three main design objectives: (a) to improve on the usability of its predecessor; (b) to provide for multilingual access; and (c) to be platform independent. To achieve these objectives, SUITEDasher employs a minimal graphical user interface, and incorporates a trigram-based probabilistic model from Dasher – a text-entry interface driven by pointing gestures. Multilingual access is achieved through a set of syntactic, lexical, and (potentially) phonetic models that may be loaded dynamically at run time. Finally, SUITEDasher is being implemented in Java as a cross-platform (Windows, Mac OS X, and Linux) application. While refining the design, we have developed several prototypes, the latest of which has been implemented in Visual Basic and C++ on the Windows platform. Preliminary usability results indicate that, in terms of typing speed, SUITEDasher appears to be 30% faster than its predecessor, and as fast as Dasher.

Keywords: Universal access, motor impairments, speech user interfaces, listening keyboard, usability evaluation.

1 Introduction

Several studies have been performed exploring the effectiveness of speech as an alternative to the physical keyboard for alphanumeric data entry. Murray et al. (1983) found that, in the context of controlling cursor movement, keyboard input is twice as fast and more preferred by users compared to speech. For editing text, Morrison et al. (1984) discovered that typists preferred the keyboard, whereas non-typists initially preferred speech but eventually switched to the keyboard. Gould et al. (1983) studied whether keyboard input could be replaced completely by speech input. In this study, a skilled typist simulated a "listening typewriter". Subjects "typed" more slowly through speech, however they liked the "listening typewriter" concept. All these studies employed speech at the level of complete words (e.g., "copy") as opposed to individual keystrokes (e.g., "c", "o", "p", "y").

Manaris et al. (2002) discovered that, in the context of arbitrary alphanumeric keystrokes, speech interaction with a simulated "listening typewriter" is better for users with motor impairments than handstick/mouthstick.[1] This is true in terms of task completion time (37% better), typing rate (74% better), and error rates (63% better). They also found that, although handstick is much worse than a QWERTY keyboard, it is approximately equivalent to conventional modes of alphanumeric input available on mobile devices (e.g., PDAs, cellular phones, and personal organizers). These modes include miniaturized keyboards, stylus "soft" keyboards, cellular phone numberpads, and handwriting recognition software. Therefore, ideal speech input is better than these modes for arbitrary alphanumeric data entry. This study contributed to the development of SUITEKeys – a speech user interface for arbitrary keyboard and mouse access (SUITEKeys 2003).

Vertanen (2004) states that, given the state-of-the-art in speech recognition, speech user interfaces have to address two usability issues: (a) poor recognition accuracy, and (b) inefficient error recovery mechanisms. To address these issues, he incorporated a speech recognition engine into Dasher – a text-entry interface driven by pointing gestures (Dasher Inference Group 2004). Our approach is slightly different: instead of adding a speech recognizer to Dasher, we incorporated Dasher's probabilistic model into SUITEKeys.

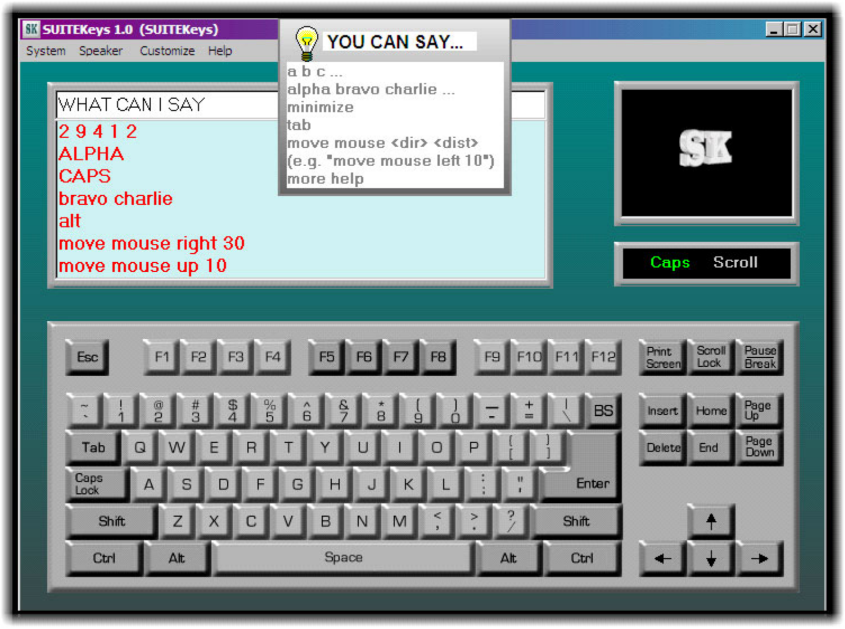
This paper presents the design of SUITEDasher – a multilingual speech user interface for motor-impaired users. SUITEDasher's architecture is based on SUITEKeys. It incorporates a trigram-based probabilistic model to

[1] Mouthstick and handstick are devices used by motor-impaired users to manipulate QWERTY keyboards.

facilitate error correction. It is being implemented in Java to be cross-platform (Windows, Mac OS X, and Linux). While refining the design, we have developed several prototypes, the latest of which has been implemented in Visual Basic and C++ on the Windows platform.

1.1 SUITEKeys

The first component, SUITEKeys, is a continuous speech application that allows users to control a virtual keyboard and mouse via speech (Manaris & Harkreader, 1998; Manaris, McCauley & MacGyvers, 2001). Instead of dictating complete words, a SUITEKeys user speaks keystrokes (e.g., 'a', 'b', 'function one'). Similarly, the user controls the mouse through spoken commands (e.g., "move mouse left", "stop"). These virtual keyboard and mouse actions are inserted into the operating system event queue. Thus, applications are not aware that these events originated from a microphone as opposed to the physical keyboard and mouse. SUITEKeys displays a virtual keyboard on the screen (Figure 1), which may be minimized for screen real-estate purposes. The architecture incorporates several components, including a dialog manager, a speech engine, a natural language model employing a left-to-right, top-down, non-deterministic parser for ambiguity handling at the semantic level, and a code generator.


SUITEKeys 1.0 (SUITEKeys)
System Speaker Customize Help
SK
Caps Scroll
Esc F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 Print Screen Scroll Lock Pause Break
Insert Home Page Up Delete End Page Down
Tab Q W E R T Y U I O P
Caps Lock A S D F G H J K L Enter
Shift Z X C V B N M Shift
Ctrl Alt Space Alt Ctrl

WHAT CAN I SAY
2 9 4 1 2
ALPHA
CAPS
bravo charlie
alt
move mouse right 30
move mouse up 10

YOU CAN SAY...
a b c ...
alpha bravo charlie ...
minimize
tab
move mouse <dir> <dist>
(e.g. "move mouse left 10")
more help

Figure 1: SUITEKeys maximized view after user has uttered, "What can I say?" to get help about the current dialog state.

1.2 Dasher

The second component, Dasher, is an efficient text-entry interface, driven by natural continuous pointing gestures (Ward, Blackwell, & MacKay 2000; Ward & MacKay, 2002). It also allows users to enter keystrokes without the keyboard . Using only the physical mouse or an eye-tracker as input, a user can write a word one letter at a time by clicking on a graphical user interface (Figure 2). This interface displays characters according to their probability of occurrence, given what has been "typed" already. Characters that are more probable are given more screen real estate to facilitate selection by the user. As predicted by Fitt's law, this mechanism improves data entry speed while reducing error rate. These probabilities are determined by the corpus, and are updated automatically based on user input. Dasher has been developed in C++ and runs on a variety of platforms including Windows, Mac OS X, Linux, and various PDAs. A significant disadvantage of the Dasher user interface is that it takes up a lot of screen area, and requires the user to cut & paste from its buffer into the application of his/her choice.

2 SUITEDasher Design

SUITEDasher integrates components from SUITEKeys and Dasher to provide multilingual, multimodal keyboard and mouse control for motor-impaired users. This approach introduces unique possibilities as well as challenges.

For instance, it is expected to increase SUITEKeys' rate of input and accuracy. Also, the combined architecture allows the user to switch at runtime between any of the available language models. In terms of challenges, issues include (a) how to combine the two probabilistic models, the speech engine's and Dasher's; (b) how to facilitate user error recovery; and (c) how to effectively expand/adapt the speech engine's phonetic model for multilingual support. SUITEKeys was written in Visual Basic, C++ and MS SAPI; SUITEDasher is being rewritten in Java incorporating CMU's Sphinx open-source speech engine (CMU Sphinx Group, 2004).

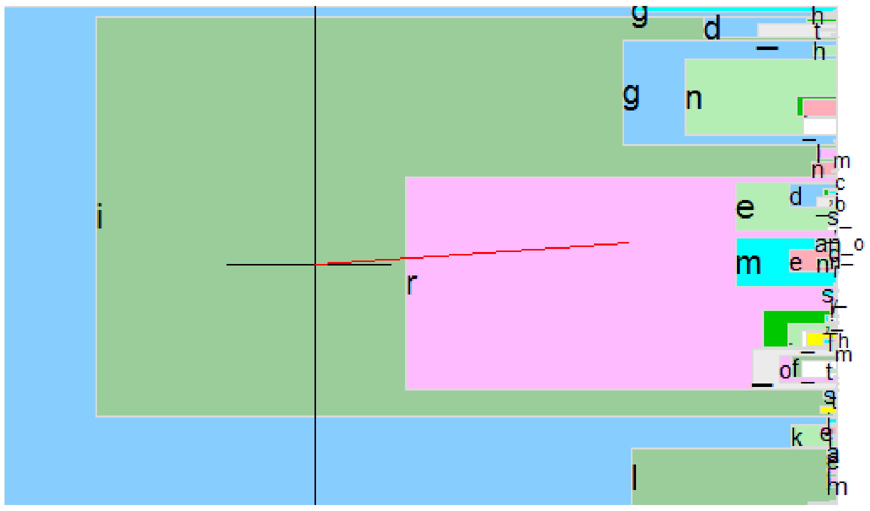


Figure 2. Dasher screenshot while user is writing the word "impaired"; alternative words easily accessible from this state include "impairment", "impairs" and "impair_".

2.1 User Interface

The SUITEDasher user interface design is informed from evaluating SUITEKeys and Dasher. The SUITEKeys interface is described extensively in (Manaris, McCauley, & MacGyvers, 2001). The Dasher interface is described in (Garrett, Ward, Murray & Cowans, 2003).

2.1.1 Graphical User Interface

In terms of the graphical user interface, SUITEDasher tries to minimize its use of screen real estate – a problem with both SUITEKeys and Dasher. Our main guideline was "less is more". This originated from the realization that, in order for the tool to be effective, it should be as unobtrusive and/or attention-grabbing as possible. As a result, there is no a virtual keyboard view as in SUITEKeys (Figure 1), nor a large graphical navigation space as in Dasher (Figure 2).

The visual presence of SUITEDasher consists of the following components:

- a pop-up menu attached to the cursor, which displays the most probable alternatives for what the user has entered (Figure 3);
- the cursor, which changes color to provide feedback from the speech engine: *white* means the speech engine is ready, *green* means the microphone is picking up speech, *yellow* means the speech engine is processing speech, and *red* means the speech engine did not recognize the last utterance.
- a system-tray icon, which displays the state of the speech engine through color: green for listening, blue for sleeping, and red for off; and
- a pop-up menu, which may be access either through speech (via "SUITEDasher" spoken command), or through clicking the mouse on the tray icon (Figure 4).

2.1.2 Speech User Interface

Similarly to SUITEKeys, SUITEDasher's architecture integrates a continuous, speaker-independent recognition engine with natural language processing components. It makes extensive use of dialog management to improve recognition accuracy. It processes input that originates as speech events and converts it to operating system keyboard and mouse events. The complete system runs on top of the operating system like any other application. Other applications are unaware that keyboard/mouse events originated as speech input.

In terms of the speech user interface, the user interface is similar to SUITEKeys (SUITEKeys 2005). The possible commands and their syntax are summarized below in English. Multilingual access is described in a subsequent section.

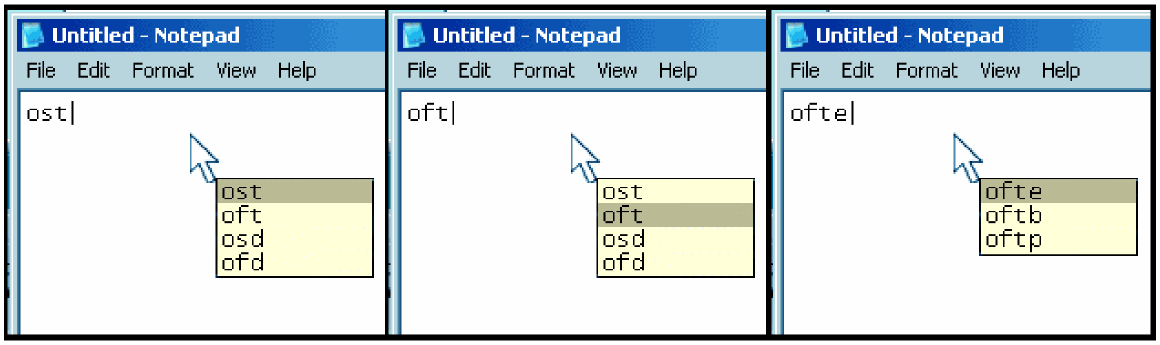


Figure 3: SUITEDasher consecutive screenshots while user is writing the word "often". (1) User says "o-f-t", but system recognizes "o-s-t". (2) User says "other" to select "oft" from the alternatives. (3) User says "e".

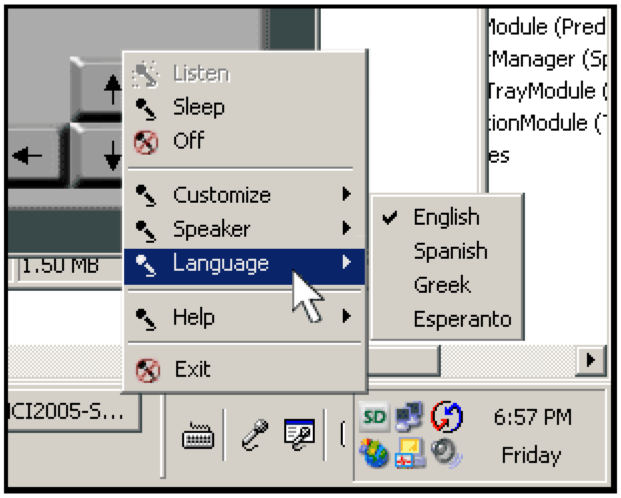


Figure 4: SUITEDasher system menu with multilingual access submenu visible.

2.1.2.1 Keyboard Speech Commands

The SUITEDasher user enters keystrokes using spoken language. Saying the name of a key (e.g., "a", "8", "function eleven", "backspace", etc.) enters the corresponding keystroke in the operating system event queue. Thus, uttered keys have the same effect as keys entered through the physical keyboard. The linguistic model includes both regular and military pronunciations. Normally, modifier keys are held pressed while typing alphanumeric keys to provide semantic variations. Saying "press" and a modifier key (i.e., "alt", "control", "shift") holds the key pressed.

Saying “release” and a modifier key, releases the key. In terms of toggle keys, saying “caps lock”, or “scroll lock” toggles the corresponding key.

2.1.2.2 Mouse Speech Commands

SUITEDasher allows users to manipulate a two-button mouse using spoken language, much as if they were transcribing to someone how to perform the same actions with a physical mouse. Specifically, the mouse may be moved in four directions: up, down, left, and right; its buttons may be clicked, double-clicked, pressed, or released.

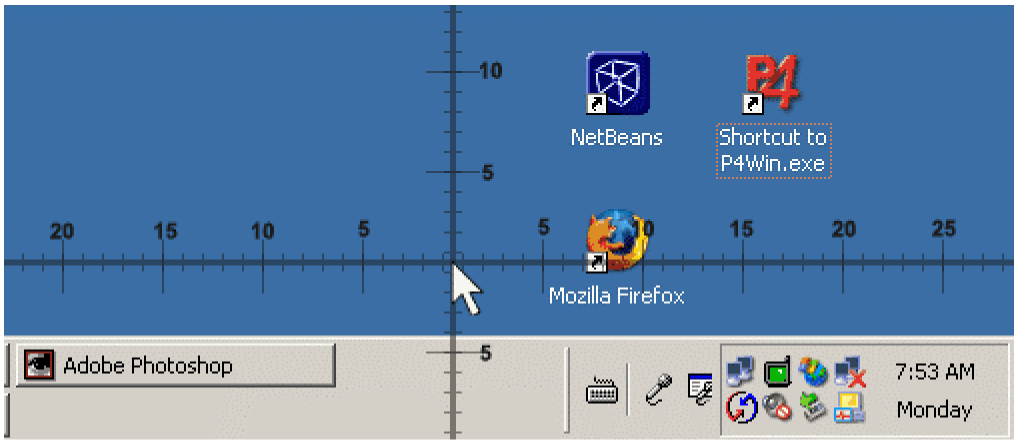


Figure 5: Mouse grid interface.

Users may reposition the mouse pointer a number of units from its original position. When the user says “mouse,” a set crosshair lines is displayed for navigation reference (Figure 5). These lines contain numbers, which indicate how far the cursor will travel on a certain direction when commanded by the user.   For example, saying “mouse down” will reposition the cursor one crosshair unit below its original position; saying “mouse right nine times” will reposition the pointer 9 units to the right (and, in the case of Figure 5, on top of the Mozilla Firefox icon).

Mouse buttons may be controlled similarly to keyboard keys. For example, saying “click left button” or “double-click right button” will perform the corresponding actions. Dragging is achieved by pressing the left mouse button (e.g., “press left button”) and moving the mouse. Dropping is achieved by releasing the left mouse button (e.g., “release left button”). The complete list of available commands

2.1.3 System Commands

In addition to the above, SUITEDasher understands the following:

- “other” or “other *x* times”, where *x* is a number, selects from the alternative error-correction menu attached to the cursor. For consistency, the suffix “*x* times” is also available for mouse movement and repeated keystrokes (Figure 3).
- “SUITEDasher” brings up the system menu (Figure 4). Any menu item with a microphone image can be selected via speech. Items with a crossed-out microphone image are not selectable via speech for othrogonality reasons. This is because these selections turn off the speech engine, and a motor-impaired user would not be able to undo these actions via speech. These items are selectable only through the mouse.

2.2 Predictive Model

Dasher incorporates a probabilistic predictive model. Given what the user has already entered, this model generates an N-Best list of letters that may possibly follow. Each letter is associated with its probability of occurrence, e.g., ‘h’ is 0.47, ‘t’ is 0.23, ‘a’ is 0.05, etc. Dasher’s predictive model is constructed from a training corpus provided by the user and/or through collecting statistics during use. Dasher learns correlations between groups of letters up to six-characters long.

The Sphinx speech engine uses a probabilistic predictive model similar to Dasher's. This model generates an N-Best list of recognized utterances. Each utterance is associated with its probability of occurrence, e.g., 'this' is 0.25, 'that' is 0.19, 'it' is 0.08%, etc. The difference between Dasher's and Sphinx's language models is that Dasher does a statistical analysis of each letter, whereas the Sphinx speech engine works at the word level.

SUITEDasher combines the two probabilistic models to improve its accuracy. Since, in the context of SUITEDasher, the "words" recognized by Sphinx are mostly alphabets, the two models are compatible. Given the N-Best structures returned by Dasher and Sphinx, we use the following formula:

$$Ci = Si * Ws + Di * Wd \qquad (1)$$

where, *Ci* is the combined probability that word i was entered, *Si* is Sphinx's probability for that word, *Ws* is the weight we place on Sphinx's prediction, *Di* is Dasher's probability for that word, and *Wd* is the weight we place on Dasher's prediction.

For example, let's assume the user has entered "an" so far. Then (s)he speaks the next letter. Table 1 displays sample N-Best lists generated by Sphinx and Dasher, respectively, for this input. The Sphinx column demonstrates how difficult it is for state-of-the-art speech engines to disambiguate similarly sounding letters (e.g., 'd' and 't'). The SUITEDasher column shows the combined probabilities according to equation (1). Assuming equal weights (*Ws* and *Wd*) of 0.5, the system would decide that the user entered 'd' with confidence of 0.45.

Table 1. SUITEDasher combined N-Best probabilities (assumes equal weights).

Sphinx (S)	Dasher (D)	SUITEDasher (C)
't' = 0.32	'd' = 0.62	'd' = 0.45
'd' = 0.28	space = 0.12	't' = 0.19
'p' = 0.19	'y' = 0.10	'p' = 0.095
'b' = 0.16	'o' = 0.07	'b' = 0.08
'g' = 0.05	't' = 0.06	space = 0.06
	'g' = 0.03	'y' = 0.05
		'g' = 0.04
		'o' = 0.035

2.3 Error Recovery

There are two possibilities for user error: either (a) the user misspoke, or (b) the system misrecognized what the user said. In the first case, the user may say, "backspace", and then enter the correct letter. In the second case, the user may select alternatives from the displayed list (Figure 3).

There is a significant difference between how Dasher and SUITEDasher employ probabilities. Dasher uses trigrams to predict the future, i.e., what letters could probably follow from what has been typed up to this point (Figure 2). SUITEDasher, on the other hand, uses trigrams to help disambiguate what has been typed already by providing alternatives to the user (Figure 3). The usability effect is similar to the T9 interface found in modern cell phones. In particular, mobile phone users may type words on their phones by pressing a sequence of numbers that correspond to each letter (Hasselgren, Montnemery, Nugues & Svensson, 2003). The system then matches the key sequence to all of the possible words in a dictionary. Although a key (number) stands for several characters, given a sequence of key taps, only a small number of actual character sequences may correspond to them. For example, if the user taps "2 2 6 8", the mobile phone displays character sequences such as "cant," "abou," "abov," and "acou." Note that all of these sequences are either words or prefixes of words found in the active dictionary.

2.3.1 Dictionary or Trigrams?

In the case of the SUITEDasher probabilistic model, we considered two choices: (a) a dictionary and (b) character trigrams. The advantage of a dictionary model is that it constraints alternatives to prefixes of known words. This is also a disadvantage, since the user may wish to type, say, a Java program: variables names would not be found in the dictionary. Obviously, the dictionary could be dynamically updated, however this introduces a trade-off between ease-of-update and pollution from erroneous additions. Since there are no probabilities associated with words, an erroneous addition has the same significance as other words found in the dictionary and quickly “surfaces” during interaction. To address this problem, dictionary-based mobile phones allow dynamic updates, but make them rather hard to do, thus reducing potential for dictionary pollution.

On the other hand, the advantage of a character-trigram model is that it provides for easy entry of non-dictionary “words”; it also provides for automatic update of trigrams during interaction: as the user types, the trigram model gets updated. If the user uses backspace to delete some characters, the trigram model gets updated. Even if the user makes mistakes, these do not surface easily, unless they are repeated many times.

A disadvantage of the trigram model is that it may generate non-words as potential alternatives of user input (Figure 3 above). However, many of these non-words have very low associated probability and, thus, “sink” below the visible part of the list (SUITEDasher displays only the seven top-most alternatives). Additionally, if the user is trying to spell an uncommon last name, or a user-defined variable name, then the trigram model may rank these non-words considerably high. This is because such non-words usually include trigrams that are common in the language model. For example, let’s assume the user is trying to spell “spaceIterator”, which is a non-word but a reasonable variable name. This input is very likely to be found higher in the list than “xzysdtjbdsf”. This is because the trigrams found in the first one (e.g., “spa”, “pac”, “ace”, etc.) are collectively more common than the trigrams in the second one (e.g., “xzy”, “zys”, ysd, etc.).

Currently, we use a trigram-based probabilistic model. However, we are also investigating a word-bigram dictionary model, as described in (Hasselgren, Montnemery, Nugues & Svensson, 2003), and its effect on SUITEDasher usability, as a supplement to our current statistical model.

2.3.2 Ambiguity Resolution

SUITEDasher’s linguistic domain may contain many ambiguous lexemes, i.e., input tokens that sound alike. For example, in English, the speech engine may easily confuse “t” for “d”, “d” for “t”, “b” for “p”, “a” for “8” or “k”, etc. As mentioned earlier, SUITEDasher displays a list of “homonym” (sound-alike) alternatives (Figure 3 above). Each alternative comprises a permutation of all possible letters that sound alike for each a recognized letter. This list is constructed from the speech engine’s N-Best list in conjunction with the trigram-based probabilistic model. It is ordered by combining trigram probabilities (see Equation 1).

The system displays the most probable alternative at the top of the list and also sends the corresponding keystrokes to the operating system event queue (in effect simulating keyboard typing). As the user enters more letters, the list is dynamically updated. It should be noted that if the top-most alternative changes, so does the word currently being typed. (When this happens, the system sends the right amount of backspaces to the event queue, followed by the keystrokes comprising the new alternative.) This allows the user to keep entering letters even if the engine has made an early mistake, since the correct alternative may rise to the top of the list and automatically update what has been typed.

At any point, the user may select from the alternative list by saying, “other”, or “other *x* times”, where *x* is between 1 and number of alternatives minus one (the first alternative is the one currently recognized). Saying “backspace” deletes one letter and, again, dynamically updates the list of alternatives. Since the list stays populated until after the end of a “word” (after a delimiter character is entered), the user may wait until then to select an alternative, if needed. The list is reinitialized when a new word begins (i.e., when the first non-delimiter character is enter after the last delimiter). Thus, error recover is mainly achieved through a combination of “backspace” and “other” commands. Finally, if the system’s N-Best list does not contain the user’s choice, the user may still enter a letter using (a) the military alphabet equivalent, or (b) the keyboard (assuming, of course, the user has some mobility).

2.4 Multilingual Access

Dasher incorporates alphabets and probabilistic models from over 60 languages, including English, French, German, Italian, Greek, and Esperanto (Dasher Inference Group, 2004). Similarly, SUITEDasher provides for multilingual access. The semantic model is the same for all languages. This model consists of three objects: a speech keyboard, a speech mouse, and SUITEDasher, and the actions each of these affords. The SUITEDasher architecture is decoupled from the syntactic and lexical models. These can be loaded dynamically at runtime (Figure 4).

Our current prototype implements only the English language. Given the simplicity of the linguistic domain, languages may share the same syntactic model. For example, the following syntax rule

<Mouse-Command> := <Mouse> <Direction> <x> <Times>

works well in English, Spanish, Greek, and Esperanto (e.g., "mouse left ten times", "ratón izquierda diez veces", "ποντίκι αριστερά δέκα φορές", and "muso maldekstre dek foje". Alternatively, different syntactic models may be loaded dynamically, as required. The same applies to lexical models.

Ideally, we could provide independent phonetic models, but this is beyond the scope of our project. This deficiency may be handled as follows: All freeware speech engines (accessible to us) provide an English phonetic model. To effect SUITEDasher recognition of non-English lexemes, we provide a phonetic transcription of each lexeme in the English phonetic model.[2] At runtime, a user-initiated change in language loads the corresponding phonetic, lexical, syntactic, and probabilistic models. This also may update the labels of the graphical user interface.

3 Usability Evaluation

Manaris et al. (2003) compare several methods of alphanumeric data entry for motor-impaired users, including handstick and ideal speech. We conducted a similar usability experiment to evaluate our SUITEDasher design (as implemented in our Visual Basic prototype), relative to SUITEKeys, Dasher, Handstick, and ideal Speech (implemented through a human transcriber in a Wizard-of-Oz setup). Our hypothesis was that SUITEDasher would perform better than SUITEKeys, due to its trigram probability model and displayed list of alternatives.

In this preliminary experiment, an expert, able-bodied user entered the following 38-word paragraph using SUITEKeys, Dasher, and SUITEDasher:

```
The language PLANNER, referred to here, is an AI language whose principal feature
is that some of the operations necessary for problem reduction are built in--
namely, the recursive process of creating a tree of subgoals, subsubgoals, etc.
```

This paragraph was used in the Manaris et al. (2003) study. Figure 6 shows the time in seconds it took to enter the above paragraph for each of the conditions.

The different input modalities ranked as follows in terms of (rounded) words-per-minutes: SUITEKeys 3 wpm (or 11:22 mins); SUITEDasher 5 wpm (or 7:59 mins); Dasher 5 wpm (or 7:25 mins); Handstick 7 wpm (or 5:21 mins); and ideal Speech 11 wpm (or 3:22 mins). In terms of completion rate all scored 100% with the exception of Dasher which scored 96.89% (due to 8 data-entry errors that went unnoticed and thus uncorrected).

3.1 Discussion

SUITEDasher typing rate improves over SUITEKeys by about 30% and is approximate to that of Dasher. This is not surprising as SUITEDasher incorporates Dasher's probabilistic model. Our preliminary results suggest that SUITEDasher's graphical user interface may be less error-prone compared to Dasher (given Dasher's less than perfect error rate, i.e., 96.89%). We expect both Dasher and SUITEDasher rates to improve further with the use of a task-specific corpus to derive the probabilistic model; for the experiment above, we used a generic corpus that came with Dasher. For example, if the user wants to write a Java program, (s)he should load a corpus from similar Java programs into SUITEDasher to improve the probabilistic model, and thus the typing rate. Also, SUITEDasher's speech engine provides for user training to improve recognition.

4 Conclusion

We presented the design of SUITEDasher, a multilingual, open-source, cross-platform speech user interface for keyboard and mouse access. We discussed how this design improves on its predecessor, SUITEKeys, due to the incorporation of the probabilistic model used in Dasher. Although far from the ideal Speech data-entry rate of 11 wpm, SUITEDasher's rate of 5 wpm is a 30% improvement over SUITEKeys.

[2] Obviously this is not ideal, and needs to be evaluated in terms of effectiveness/usability.

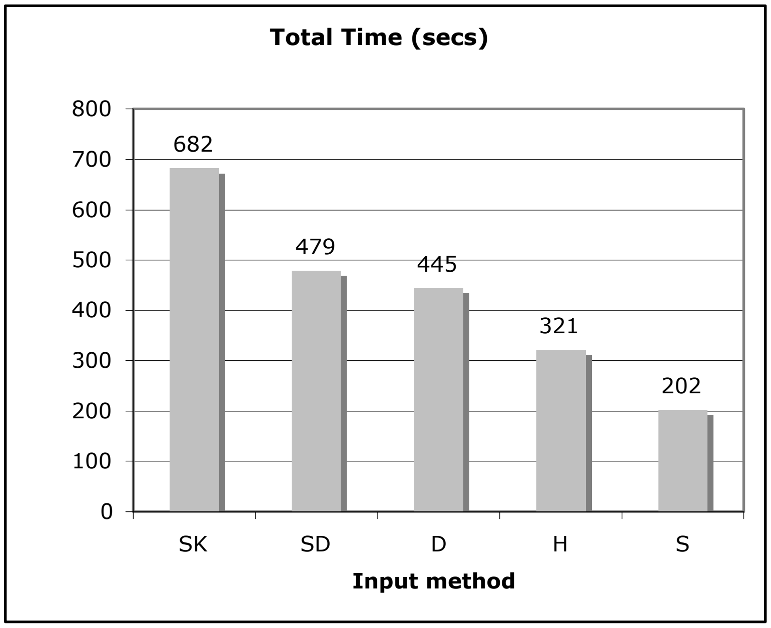


Figure 6: Total time required to enter benchmark paragraph of 38 words using SUITEKeys (SK), SUITEDasher (SD), Dasher (D), Handstick (H), and ideal Speech (S).

Currently SUITEDasher performs character-trigram analysis to predict probable alternatives to what was recognized by the speech engine. We are exploring ways to supplement this analysis with a word-bigram dictionary in order to improve the application's usability and thus increase its data-entry rate. We are also exploring different capabilities of the CMU Sphinx speech engine, such as modifying the phonetic model to cater to other languages.

Acknowledgements

The authors wish to acknowledge Wayne Staats & Luca Pellicoro for their contribution in the SUITEDasher probabilistic model design and evaluation; and Yoshiyuki Mihara for his contribution to the design of the mouse grid interface.

References

CMU Sphinx Group (2004), Open Source Speech Recognition Engines.
Retrieved September 28, 2004, from http://cmusphinx.sourceforge.net.

Dasher Inference Group (2004). Dasher Project: Languages.
Retrieved July 14, 2004, from http://www.inference.phy.cam.ac.uk/dasher/Languages.html.

Garrett, M., Ward, D., Murray, I., & Cowans, P. (2003). Implementation of Dasher, an Information Efficient Input Mechanism. *Proceedings of the 4th Annual GNOME Users And Developers European Conference.*
Retrieved from http://www.codon.org.uk/~mjg59/guadec.pdf.

Gould, J. D., Conti, J., and Hovanyecz, T. (1983). Composing letters with a simulated listening typewriter. *Communications of the ACM*, *26*, 4:295-308.

Hasselgren, J., Montnemery, E., Nugues P., & Svensson M. (2003). A Predictive Text Entry Method Using Bigrams. *Proceedings of the Workshop on Language Modeling for Text Entry Methods, 10th Conference of the European Chapter of the Association of Computational Linguistics*, Budapest, Hungary, pp. 43-49

Manaris, B., & Harkreader, A. (1998). SUITEKeys: A Speech Understanding Interface for the Motor-Control Challenged. *Proceedings of The Third International ACM Conference on Assistive Technologies (ASSETS '98)*, Marina del Ray, CA, pp. 108-115.

Manaris, B., MacGyvers, V., & Lagoudakis, M. (2002). A Listening Keyboard for Users with Motor Impairments—A Usability Study. *International Journal of Speech Technology*, 5(4), 371-388.

Manaris, B., McCauley, R., and MacGyvers, V. (2001). An intelligent interface for keyboard and mouse control – Providing full access to PC functionality via speech. *Proceedings of 14th International Florida AI Research Symposium (FLAIRS-01)*, pp. 182-188.

Morrison, D. L., Green, T. R. G., Shaw, A. C., and Payne, S. J. (1984). Speech-controlled text-editing: Effects of input modality and of command structure. *International Journal of Man-Machine Studies*, *21*, 1:49-63.

Murray, J. T., Van Praag, J., and Gilfoil, D. (1983). Voice versus keyboard control of cursor motion. *Proceedings of the 27th Annual Meeting of Human Factors Society* (p. 103). Santa Monica, CA: Human Factors Society. (cited in Shneiderman, 1997).

Shneiderman, B. (1997). *Designing the User Interface*, 3rd ed. Reading, MA: Addison-Wesley.

SUITEKeys (2005). SUITEKeys v.1.0 for MS Windows.
Retrieved February 20, 2005, from http://www.suitekeys.org.

Vertanen, K. (2004). Efficient Computer Interfaces Using Continuous Gestures, Language Models, and Speech, *Masters of Philosophy Thesis*, University of Cambridge.
Retrieved from http://www.keithv.com/papers/Efficient_computer_interfaces.pdf.

Ward, D., Blackwell, A., & MacKay, D. (2000). Dasher - a Data Entry Interface Using Continuous Gestures and Language Models. *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology (UIST 2000)*, San Diego, CA, pp. 129-137.

Ward, D., & MacKay, D. (2002). Fast Hands-free Writing by Gaze Direction. *Nature* 418, 838-840.