

Implementation of Dasher, an information efficient input mechanism

Matthew Garrett David Ward Iain Murray Phil Cowans
David Mackay
Cavendish Laboratory, Madingley Road, Cambridge, CB3 0HE, UK

July 11, 2003

Abstract

Dasher is a novel information efficient text entry system using a language model to offer predictions to the user without constraining the range of words which can be written. In this paper, we discuss the implementation of Dasher and how it is well suited to providing input on keyboardless devices and for disabled users. We will also look at how its integration with the GNOME accessibility framework allows Dasher to be not just a text entry system, but a comprehensive application control system.

1 Introduction

Conventional keyboards provide an effective means of data entry to computers. However, they are inappropriate in at least two cases - where the device is insufficiently large or where the user cannot use a standard keyboard.

Dasher uses a dynamic display and an adaptive language model to provide an efficient means of text input that is not dependent upon a physical keyboard or full mobility. In combination with the GNOME accessibility framework, it provides a mechanism for full control of a computer system.

2 A description of Dasher

The standard configuration of Dasher provides a rectangular display with 27 boxes on the right hand side of the screen representing the 26 letters of the Roman alphabet and a space character (see Figure 1). The user chooses a character by making a gesture towards it, which causes the display to zoom in towards the chosen point. As the character moves towards the centre of the screen, more characters appear within each rectangle which represent possible continuations of the chosen text. For example, a user wishing to type “the” would move towards the “t” character, then the “h” character within it, finally choosing the “e” character that is within the “h” character.

The height of each rectangle (and therefore the ease with which it may be picked) is proportional to the probability of that character being chosen given the characters that have been chosen so far. As an obvious example, the “u” character within a “q” character will be very large in English. As a result, the most probably phrases are the easiest to enter.

This representation of the characters is analogous to arithmetic coding. The right hand edge of the screen represents a probability continuum from 0 to 1, and is subdivided according to the probability of each letter being chosen with a small

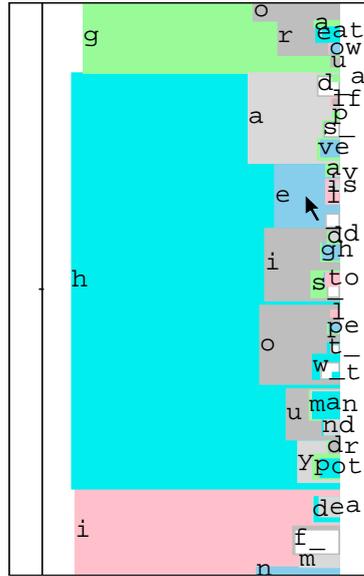


Figure 1: A user entering the word “hello”. Note the alternative predictions available to choose from, such as the beginning of “history” and “human”

“fudge” factor to ensure that each character is above a minimum size. We can think of all possible strings being arranged in alphabetical order, each occupying an amount of the initial screen proportional to its initial probability. The user simply zooms in on the tiny section of the screen corresponding to the desired string.

A language model determines the probabilities using the PPM algorithm (Bell et al., 1990). PPM assigns the final probability of a character by blending the predictions based on the previous 1, 2, 3, 4 and 5 characters in order to ensure that reasonable predictions are made even if the 5 character string has never been previously encountered.

3 Methods of driving Dasher

Dasher is reliant on the user being able to indicate their desire to move in a given direction. This may be achieved in several ways.

1. Mouse

On a desktop, mouse input is the standard method for using Dasher. The position of the mouse pointer on the canvas represents the point that the user wishes to be at a short period of time in the future. Starting and stopping Dasher is achieved by clicking the mouse button or another key if preferred (Ward and MacKay, 2002, discusses speed of entry obtained over time).

2. Pen

Pen input functions in a similar fashion to mouse input, with the position of the pen indicating the user’s desired destination. Dasher moves when the pen is in contact with the screen, and stops when the pen is released. This method is best suited to handheld devices.

3. Eyetracking

An eyetracker can be used to control the mouse position. We find it is helpful to modify the mapping of the coordinates from the screen to the Dasher model

may be modified in order to allow the user to look further ahead without having to run at full speed, and so that extreme upward and downward mouse coordinates cause a reduction in the speed of the zooming.

4. Head mouse

The Naturalpoint headmouse uses a small reflective dot attached to the user's head in some fashion and a camera that picks up reflections. This method works in much the same way as mouse input.

5. Keyboard or switch

All the above mechanisms require some degree of fine motor control or the ability to keep the head steady enough to operate an eyetracker. An alternative is to have a group of fixed points that the user may select, and keyboard or switch input to select among them. This may be timing dependent (ie, the point to be chosen cycles and pressing a switch chooses it), or timing independent (ie, each point is chosen by a different switch or one switch is used to choose the point and the other confirms the selection).

4 Advantages of Dasher over traditional on-screen keyboards

Due to the way in which Dasher zooms towards the desired character, it is unnecessary for the user to choose a precise area of the screen. As the area containing the desired character zooms in, the target becomes larger and so the user can correct their input in order to select the desired character. In this way, Dasher can be compared to driving a car – the “driver” is not required to make precise inputs, as they can be corrected afterwards. As a result of this, Dasher requires less accuracy in the input mechanism and is less tiring than using an on-screen keyboard.

Dasher's predictive techniques are more integrated than traditional on-screen keyboards. Those used on PDAs usually lack any sort of prediction. More advanced examples (such as GOK) include the ability to complete words based on prediction, but this requires the user to scan the range of offered predictions and choose one requiring frequent “context switching” between tasks.

5 The implementation of Dasher

Dasher consists of a platform-independent core tied to a platform-dependent interface. The core receives information such as input location and text context from the interface, and calculates the layout of the Dasher canvas accordingly (see Figure 2). This is then passed back to the interface to render. The core is written in C++ and has a single external dependency on the Expat XML parsing library. A C interface to the core is also available. Interfaces are typically somewhere in the region of 2500 lines of code. As a result, porting Dasher to a new platform or environment is generally a fairly trivial task.

One of the consequences of this implementation is that each interface is free to collect additional context from its environment and pass this on to the core. As an additional option, an interface may pass information about “control” commands such as menu selection to the core in the form of a tree (see Figure 3). These then appear under a separate control node. When a control node is selected, rather than a character being output the interface is informed that a control event has occurred and then reacts appropriately. As the functionality offered by the control mode is

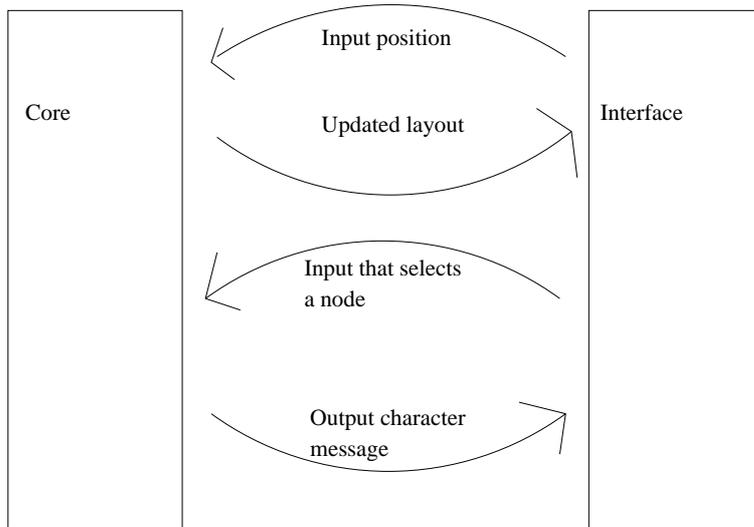


Figure 2: The interaction between the core and the interface

inherently interface specific, this allows tight integration between Dasher and its host environment without compromising its portability.

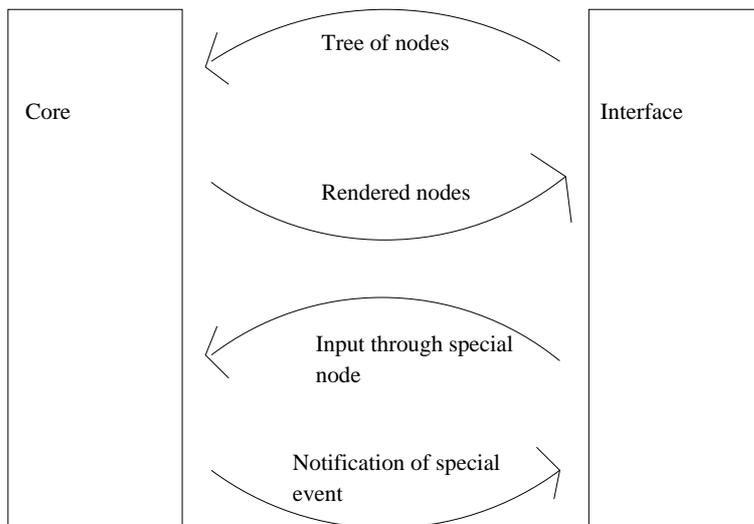


Figure 3: The interface passes a tree of special nodes to the core, which then causes them to be rendered. When one is selected, the core sends an event message to the interface which handles it appropriately

6 Integration of Dasher with the GNOME environment

6.1 Communication with other applications

In its initial form, Dasher simply output characters to an edit box. From here they could be saved or cut and paste into other applications – however, there was no

direct interaction with the rest of the desktop.

6.1.1 Text input to applications

There are two primary methods by which applications may send input events to other X applications.

1. `XTestFakeKeyEvent`

This X extension allows local applications to generate an event indistinguishable from a keyboard input event. Input will be sent to the window with input focus.

2. `XSendEvent`

This allows an X event to be constructed by an application and sent to an arbitrary window. The primary issue with this method is that these events are distinguishable from standard X events, and certain applications (such as `xterm` and `emacs`) will ignore them.

The `atk` library generates `XTestFakeKeyEvent`, and therefore it is necessary for input focus to be set to the desired window. Dasher therefore refuses to accept input focus when being used in this mode. The user selects the window they wish to enter text into and then start Dasher, which then does an `XGrabPointer` in order to prevent the user from accidentally focussing another window. The interface then outputs characters to the `atk` library rather than the edit box, causing a keyboard event to be faked and text entered into the desired window.

The prime drawback that currently exists is that it is still necessary for a keycode or keysym to be generated, and there is no simple way to generate these given an arbitrary character. The `at-spi` layer contains a lookup table that translates a subset of UTF8 into X keysyms, but this is still far from ideal.

6.1.2 Application control

The `at-spi` interface provides the functionality needed for applications to retrieve information about other applications and then use that information to interact with them. This allows the Dasher interface to retrieve information about the menu structure of accessible applications and pass this information as a tree of nodes to the core. When the core notifies the interface that one of these nodes has been selected, the Dasher interface can then signal this to the application. As a result, the user is then able to control the application without having to leave Dasher. The usual benefits provided by Dasher apply here also, as reduced accuracy is required in order to interact with the application's menus.

6.2 Internationalisation of Dasher

Dasher's prediction model is not specific to English, and Dasher can be trained in any language providing that a sufficient sample of that language is available. Dasher is fully UTF-8 based, and as a result adding support for any language is not a difficult task.

7 Conclusion

Dasher is a novel method for both text input and application control that avoids both the need for a physical keyboard and significant levels of mobility. It uses the accessibility features provided by the GNOME environment to enhance the functionality it offers users.

References

Bell, T., J. Cleary, and I. Witten (1990). *Text Compression*. Prentice Hall.

Ward, D. J. and D. J. C. MacKay (2002). Fast hands-free writing by gaze direction.
Nature (6900), 838.