IST-2003-511598 (NoE)

COGAIN

Communication by Gaze Interaction

Network of Excellence

Information Society Technologies

D4.1 Design specifications and guidelines for COGAIN eye-typing systems

Due date of deliverable: 31.08.2005

Actual submission date: 05.09.2005

Start date of project: 01.09.2004 Duration: 60 months

IT University of Copenhagen

Project co-funded by the European Commission within the Sixth Framework Programme (2002–2006)		
Dissemination Level		
PU	Public	x
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Hansen, J.P., Johansen, A.S., Donegan, M., MacKay, J.C., Cowans, P., Kühn, M., Bates, R., Majaranta, P., and Räihä, K.-J. (2005) ***D4.1 Design specifications and guidelines for COGAIN eye-typing systems***. Communication by Gaze Interaction (COGAIN), IST-2003-511598: Deliverable 4.1. Available at http://www.cogain.org/results/reports/COGAIN-D4.1.pdf

Main author: John Paulin Hansen (ITU)

Contributors: Anders Sewerin Johansen (ITU)
Michael Donegan (ACE)
David J.C. MacKay and Phil Cowans (UCAM)
Michael Kühn (UNI KO-LD)
Richard Bates (DMU)
Päivi Majaranta and Kari-Jouko Räihä (UTA)

Thanks to: Participants of the Ph.D. course “Under the Hood of Advanced Writing Systems”, held in conjunction with the Work Package 4 research retreat on “design of eye writing systems” at the first COGAIN camp in Copenhagen, May/June 2005, who contributed as well.

Table of Contents

Executive Summary

This deliverable defines common design guidelines and principles for eye-writing systems developed within the COGAIN network. Unified specifications and a shared terminology for gaze interaction will be established that may enable integration and exchange of modules and functionality between future eye-typing systems inside and outside the COGAIN network. Through the convergence of methodology, exchange of ideas, and software resources, the long-term goal is that the users of eye-writing systems will be able to switch between systems and use their personalised features in all of them. A coordinated effort is needed to bend the existing systems towards each other by a modular approach when assembling a communication and environmental control solution for an individual user. This deliverable discusses the usability requirements and needs assessments for eye-writing systems, and it offers recommendations for areas and software solutions that may form the bases for integration.

Members of Work Package 4 agreed on the following:

- Common usability issues identified (cf. Chapter 2) should be taken into account when developing eye-writing systems. The optimal eye-writing system presents as few targets as possible, with a function as close to user expectations as possible, ideally placed as close as possible to the current area of interest, in the expected place, which supplies unambiguous, specific feedback, preferably allowing the user to detect and correct errors easily and with a minimum of interaction. In order to support rote learning, the system must be fully predictable (or nearly so), lessening the need for frequent visual verification.
- The current forms of system architecture (cf. Chapter 3) are complex yet flexible enough to allow for the convergence. They all make it possible to interface with eye trackers at the level of standard pointer control, and they all allow for export and exchange of the language models generated. The export/exchange of language models may become a standard file format as an effect of the convergence.
- The COGAIN network opts for a strategy of convergent de-facto implementations, as none of the partners would otherwise have the resources needed to rebuild (or substantially modify) the large amount of code base in accordance with new programming specifications. The COGAIN Work Package 2 standard wrapper (see D2.2 Requirements for the common format of eye movement data by Bates et al., 2005) for interfacing to eye-tracking systems, which provides a single API to all eye trackers, might be of value here.
- Convergent de-facto implementation would include the ability to launch other eye-writing systems from within each other, and to translate one language model format to another system format, using software tools we recommend for development within the network.

1 Introduction to eye-writing systems

Augmented and Alternative Communication (AAC) is the general term for systems that enable people with physical or cognitive handicaps to communicate or operate computers or other equipment. AAC systems range from simple design modifications of the ergonomics of standard QWERTY keyboards to input systems for special symbol languages (e.g. Bliss or pictograms), with simplified user interfaces that allow for input by users who have both motor and cognitive difficulties.

Eye-writing systems are found in two basic forms: 1) systems built for AAC in general that are also well-suited to eye writing, as with standard on-screen keyboards (e.g. Point for Windows and Wivik), or 2) systems built specifically for eye writing (e.g. GazeTalk), some of which may also be suitable for AAC in general.

A primary goal of many eye-writing systems is to allow disabled people to take part in conversation—to function as a voice prosthetic. These systems are characterised by:

- Independence from keyboards as an input device for producing text.
- Production of conversation-like text ('type-to-talk') as a primary focus.
- Speed and accuracy rather than presentation as essential features.
- Multimodal interaction, especially gaze input, as a solution to the user's limitations in mobility and motor performance.

Secondary goals include the ability to compose text for writing—letters, e-mails, diaries, memoirs, and so on. System requirements would include the support of text editing (e.g. handling portions of text), saving, copying, and pasting; if possible, some kind of spell-checking would be desirable as well.

Further goals of eye-writing systems include the ability to fill out electronic forms, such as text fields in web-search tools, the names of files, or URLs.

1.1 Principles of eye writing

To use a conventional mouse, a user must first position the cursor over the relevant item before a selection can be made. Similarly with eye tracking, the item to be selected first must be focused on. Many eye-tracking devices actually emulate a conventional mouse, with the user's eyes serving as the mouse itself.

Most eye-typing systems are implemented by the use of an on-screen virtual keyboard. An eye-tracking device tracks the user's point of gaze, and a computer program analyses the gaze behaviour. From the analysis, the system decides which letter the user is focusing on and whether the user wants to type it. The process comprises the following steps. First, the user decides which letter he or she wants to type and *focuses* his or her gaze at the corresponding virtual letter on the on-screen keyboard. Typically, the system then gives *feedback* by highlighting the focused letter, for example, by placing a cursor on the virtual key. The user *confirms the selection* by continuing to fixate on the letter, thus using time as an activation command. Alternatively, the user can also trigger a switch to select the highlighted item. The system may also provide auditory feedback to indicate that a virtual key press was successful. The result of these steps is the selected letter appearing in the open text field (for more information, see Majaranta and Räihä, 2002).

1.1.1 Dwell time selections

Focus can be shifted from item to item by redirecting gaze fixation. The duration of an uninterrupted fixation on an item is called *dwell time*. Dwell times applied in various systems range from 100 to 3000 milliseconds (ms), depending on tracking performance, user skill, type of command, and cost of errors (Hansen et al., 2003). If dwelling is used for focusing, the system usually provides the user with some indication that the predefined dwell time has elapsed. The feedback methods may be acoustic (e.g. a 'click') or animated (e.g. a moving figure). Mouse emulation with dwell time is the most widely used method for selecting an item using the point of gaze to define the location of focus.

There are three different methods for implementing dwell activation (Hansen et al., 2003):

Continuous-dwell activation. A command is executed when a button is activated without interruption for a certain preset time. If the activation is terminated before that preset time, the time counter is reset. This type of dwell activation is found in most gaze-activated systems.

Accumulated-dwell activation. A command is executed when a button is activated for a certain preset time, regardless of the actual number of activations. The time (evidence) counter of the command may be reset by some local or global system event.

Adaptive-dwell activation. The dwell time (continuous or accumulated) becomes dependent on user-behaviour patterns. Patterns might be based on frequency of use, frequency of selection error, number of activations terminated by the user, or other factors.

The time remaining before activation can be indicated within the button as a progress bar or an animation sequence. Feedback can also be integrated by changing the pointer symbol or using auditory signals.

1.1.2 Scanning

Some people have difficulty fixing their gaze because of their state of health. Some cannot sustain their gaze for the duration needed to focus, while others are able to move their eyes in only one direction. In such cases, other methods for selecting an item are needed.

The eyes can be used as simple one-way or two-way switches (Majaranta and Räihä, 2002). A method called *scanning* can be used to shift the focus from one item to another. Scanning is widely used among the disabled in various AAC systems. So called 'step scanning' allows the user to use one switch to change the focus from one item to another, and another switch to select the item in focus. 'Automatic scanning' is used if the user has only one switch; the focus automatically shifts from item to item after a definable time, and the user only needs to activate the switch when the desired item (e.g. a letter) is in focus (highlighted).

Item by item scanning is quite a slow procedure. Therefore it is advantageous to scan larger groups of items before focusing on a discrete item.

1.1.3 Eye writing by navigation and searching

Dasher (Ward, 2001; Ward and MacKay, 2002) provides a completely different way of using gaze direction from dwelling and scanning. Human eyes evolved for navigation, among other things, and Dasher facilitates the writing of text by navigation through a zooming world. When the user spots the next syllable or word that he or she desires, the display zooms in on possible continuations of the sentence. When Dasher's language model does a good job, the user's brain will easily spot the continuation that is sought, and, a moment later, the continuation after that, and so on. Even when the language model does an insufficient job of predicting the required text, the user can still find the desired text by navigating Dasher's predictions in standard alphabetical order.

Dasher operates in all languages, and its predictions can be personalised to the user's writing style easily by feeding the language model with example documents. The language model learns constantly as the user writes. During the first few minutes of use, novices may find Dasher difficult to understand, but once this entry barrier is overcome, excellent communication speeds are possible. After two hours' practice, writing speeds of 25 words per minute can be achieved, and the frequency of spelling errors with Dasher is negligibly small. Most experienced users find writing with Dasher much less stressful than writing with on-screen keyboards by dwell selection.

1.2 Eye tools for editing text

In principle, one could exploit the editing features of any common text editor (such as Microsoft Word) if it could be operated by gaze. Because the accuracy of the measured point of gaze is not good enough for operating the tiny objects on a conventional computer operating system (such as Microsoft Windows), special techniques have been developed for gaze control. The most obvious solution is to magnify parts of the screen to enable selecting the small targets (Ashmore et al., 2005; Bates and Istance, 2002).

Another problem with gaze control is that many standard applications rely heavily on a mouse. Several solutions have been developed, but the most common is an on-screen palette that depicts buttons for equivalent mouse actions (left and right click, drag, scrolling). The problem here is that the context is lost. With a conventional mouse, the user moves the mouse by hand and can observe the mouse cursor and the results of the mouse actions simultaneously. With a gaze-controlled operation, however, using the separate buttons requires the user to first look at the command button (e.g. right click) and then redirect his or her gaze to the target object.

We propose an eye widget called *iPie* (Fig. 1). A 'pie menu' is a pop-up menu that appears at the place of focus. Menu items are placed in a circular pattern around the centre of the pie, and a stroke or gesture activates the command that falls in the section that lies in the direction of the gesture. Pie menus have proved faster than ordinary linear menus in normal mouse-based interaction (Kurtenbach and Buxton, 1994).

iPie is an eye-manipulated pie menu with a large hole at its centre. The menu can be shown as needed (e.g. by extended dwell time) and manipulated and positioned by gaze. iPie does not necessarily move along with the user's gaze; the user can freeze the iPie in place and use gaze for selecting its commands.

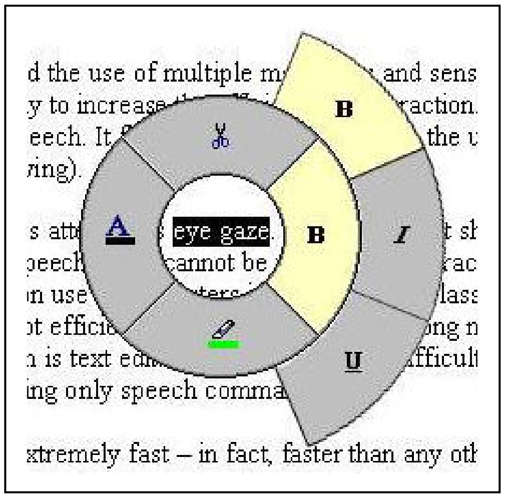


Figure 1. iPie allows the user of a gaze-interactive system to edit text by activating a widget on top of the selected text. When activated, the user unfolds rings of hierarchical functions by gazing at the relevant item on the iPie.

iPie may have the potential to solve some lingering problems related to eye typing and editing—of course, further testing is needed in these areas:

- By transferring commands into the iPie, the number of commands on the virtual keyboard decreases. More space can be devoted to important virtual keys or the text area.
- Because the iPie is context-sensitive, the needed commands are available to the user as soon as the menu is activated.
- The user does not lose the context, because there is no need to switch between the text and the virtual keyboard. The user can see the task at the same time he or she works on it because of the hole in the centre of the pie menu. (An alternative design is to make the sections partly transparent.)
- The number of commands (sections) in the iPie can be quite high if one uses the area outside the iPie—that is, the sections extend and widen beyond the visible pie. Thus, the inaccuracy of gaze should not be as much of a problem with iPie as it is using conventional menus with very small menu items.

Simple tasks can be difficult with gaze control, such as positioning a caret when one wants to edit a word inside a chunk of text. If mouse controls and other editing commands could be placed within a context-sensitive iPie menu, the user could conceivably execute actions much more easily, as the needed commands would be conveniently available near the target object.

2 User-requirement specifications

This chapter summarises and discusses some of the major points from the COGAIN deliverable 3.1 "User requirements report with observations of difficulties users are experiencing" from the application development point of view. In addition, it includes parts adapted from a work in progress by Johansen and Hansen (2005).

One of COGAIN's aims is to help to make Eye-Control Technology available and accessible to as many of those who might benefit from it as possible. This is particularly important because their other methods of access to technology might be slow, painful, harmful, or even impossible for them. Globally, there are millions of people who fall into this category. They range from some of those with conditions such as athetoid cerebral palsy to those with strokes or head injuries. With athetoid cerebral palsy, there might be strong, but uncontrolled, body and head movements. Certain people with head injuries, on the other hand, might find any form of movement, including eye movement, difficult to either initiate or control.

Eye control is a comparatively recent development for people with disabilities. From the literature and data collected it seems that, at present, eye control can only be used effectively to meet a limited range of what many potential users require of it. Furthermore, it can only be used effectively by a limited number of people with disabilities (see COGAIN D3.1 by Donegan et al., 2005 for more information).

While there are many people with disabilities who use technology successfully by one access method or another, the critical issue is the *quality* of access in relation to usability issues. While they might be able to use technology to achieve of the things they wish to *without* using Eye Control, the fact remains that Eye Control might potentially offer them *a more effective, efficient, and satisfying form of access*. Their current methods of access to technology might be slow or labour-intensive for them, whereas Eye-Control Technology might offer a much more direct and efficient form of access, at least for some of the applications they wish to use.

2.1 Usability factors in eye-writing systems

Usability may be defined as 'the effectiveness, efficiency, and satisfaction with which specified users can achieve specified goals in particular environments' (ISO/DIS 9241-11). Major dimensions of usability have been characterised by Patrick Jordan (2000) by the terms in Figure 2.

A fully predictable system, such as a static QWERTY on-screen keyboard, intuitively supports rote learning, which lessens the cognitive load for skilled users, as well as allows them to apply more advanced strategies. The goal of touch-typist training is in fact to achieve a level of rote learning that allows the typist to focus attention on the subject material to be typed, with occasional visual verification of the actual text typed.

The quality of feedback that the eye-writing system supplies on input is also very important, both in terms of the level of performance that can be achieved and the number of usage scenarios that are made possible with the device. If no feedback is given at all, the user will spend more time verifying that the intended result was achieved. If the feedback is unclear, either because it is inconsistent or disproportional to the input, it may induce lower performance than would the absence of feedback; it would be counterproductive if pressing a letter key in a word processor programme resulted in a loud, jarring sound, whose pitch was dependent on the letter typed. This principle is well known from the normal operation of current (2005) mainstream computers: mice and keyboards provide both tactile and auditory feedback when a button or key is pressed, and system messages have distinct visual and auditory cues depending on their content (tip, information, warning, severe error condition, etc.), which all help us to verify that we have operated the system as we intended.

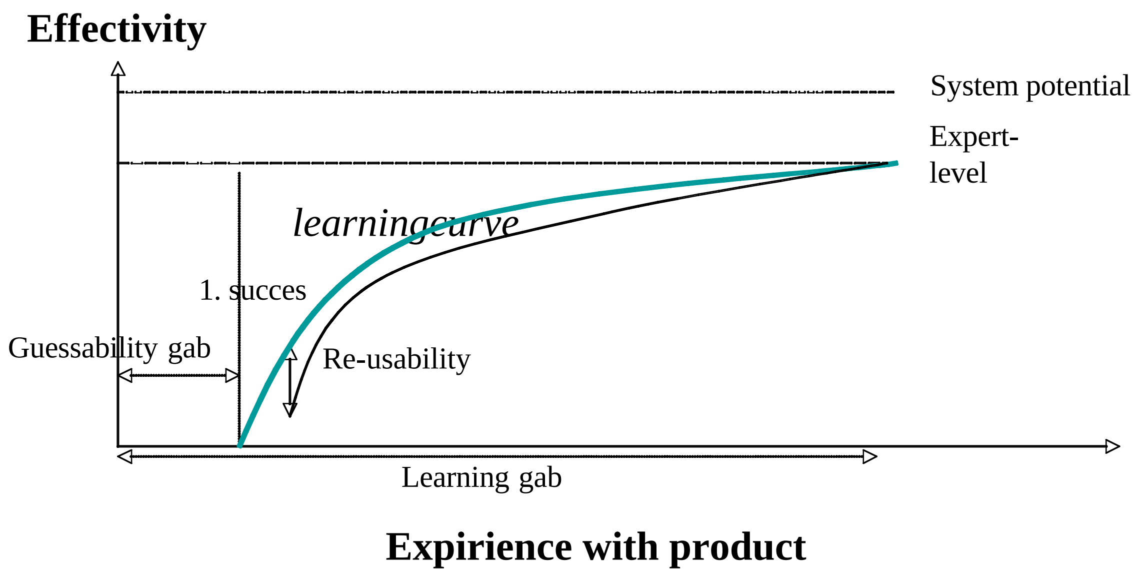


Figure 2. The major usability components of efficiency. Legend: *Guessability*—Cost of first-time use; *Learnability*—Cost of reaching a competent level of performance; *Experienced-user performance*—Efficiency level of trained user; *System potential*—Theoretical performance level; *Reusability*—Cost of being away from the system for some time.

Finally, the cost of detecting and correcting errors is a significant factor, especially when designing systems for casual or novice users who can be expected to exhibit a higher error rate than expert users. Errors can be divided into two classes: motor and cognitive. Given sufficient feedback from the system, the operator should be able to detect motor errors with some ease. Cognitive errors, however, cannot always be detected or corrected by the operator, such as in the case of dyslexic operators who enter text. Designing systems that are robust and efficient in the face of errors is thus two tasks in one, as techniques that aim to reduce or aid in recovering from motor errors are not necessarily effective when applied to cognitive errors, and vice versa.

The general conclusion is that the optimal system presents as few targets as possible, with a function as close to user expectations as possible, ideally placed as close as possible to the current area of interest, in the expected place, which supplies unambiguous, specific feedback, preferably allowing the user to detect and correct errors easily and with a minimum of interaction. In order to support rote learning, the system must be fully predictable (or nearly so), lessening the need for frequent visual verification.

2.2 Language, text, and language models in writing systems

Text is highly redundant, as can be seen from the fact that one usually achieves compression rates of >50% when compressing text with programmes such as bzip and WinZip. Although we usually store text using eight bits per character, experiments by Shannon (1951) indicated that the entropy, or the actual information content, of English text is between 0.6 and 1.2 bits per character. It follows that, on average, a very low input rate is required in order to compose text. However, this is only possible if the device used for text entry can interpret this information correctly, in effect inferring the other seven bits from general knowledge of language and context using a *language model*. Language models are used extensively in eye-writing systems to allow for selections by gazing at one key at a time. They speed up writing, but they also introduce usability problems of their own.

Language models can be based on many different algorithms, with varying results in terms of performance, capability, and resource consumption. A simple language model (frequency count) achieves an entropy of 4.03 bits per character (Shannon, 1951), whereas an advanced state-of-the-art model (maximum entropy) achieves 1.2 bits per character (Rosenfeld, 1996). The traditional statistical language modelling *n*-gram approach is relatively high-performance at 1.5 bits per character (Tilbourg, 1988). David J. Ward (2001) tabulated the performance of most current approaches and discussed the construction of language models extensively.

An alternative to the traditional *n*-gram word-level model is to run a compression algorithm in reverse, as seen in the Dasher system (Ward, 2001) and the 'reactive keyboard' (Darragh et al., 1990). In Ward (2001), this approach was implemented and evaluated. The conclusion drawn was that, although it did have several advantages—primarily, that training the model is quite a lot easier than is the case for an *n*-gram word-level model—the performance was lower than what can be achieved with the *n*-gram approach. In both cases, the 'PPM' compression algorithm was used, which, according to the tabulation in Ward (2001), performs at 2.39 bits per character, as opposed to 1.5 bits per character for an *n*-gram model. The PPM compression algorithm was described in greater detail by Ward (2001) and Darragh et al. (1990).

Most language models must be 'primed' to perform optimally. Take, for example, a simple 3-gram model, which predicts a word on the basis of the two words preceding it; the model requires priming with two words before it can supply any predictions for the third. In other words, most (possibly all) language models will supply predictions that are unreliable, or at best suboptimal, for the first few words or characters of input. Consequently, when designing text-input systems that are based on language models, one must allow for a certain error rate, even if the language model performs very well on average.

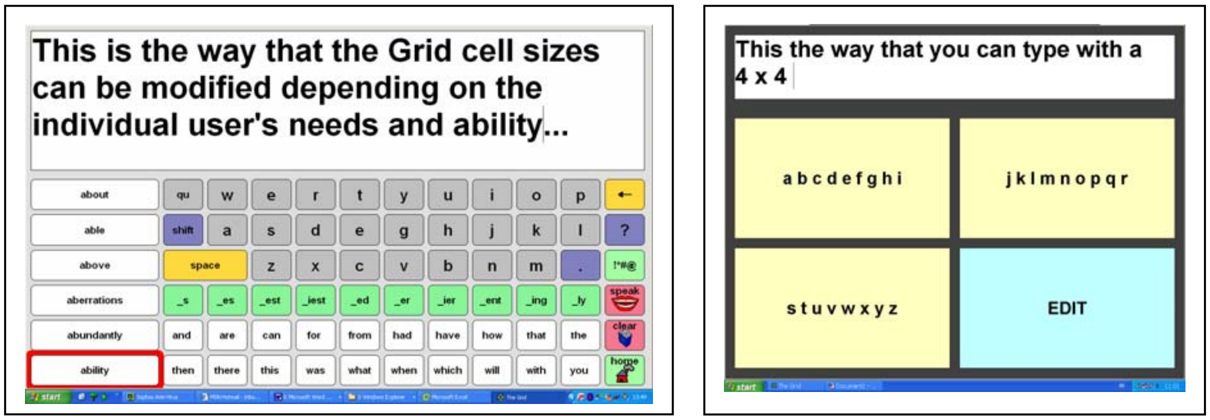
One of the most common problems when using word-level language models is the built-in dictionary: what happens when the user wants to enter a word that is not in the dictionary? It is possible to calculate the probability that a word is 'unknown', but obviously it is impossible to consistently predict the actual unknown word. It is not merely impractical to add all known words to the dictionary (*Webster's Ninth New Collegiate Dictionary* boasts almost 160,000 entries!)—but it is virtually impossible, as new words are continually added to the vocabulary of all living languages. Additionally, many countries are bilingual or even multilingual, which not only compounds the dictionary problem, but also necessitates a text-entry mode that is either not based on a language model, or that allows one to switch between or integrate several language models.

A well-known property of language models is that an in-domain model performs better than an out-of-domain model, even when the domains and the test corpus are very similar. This indicates that an adaptive language model, which integrates the user's vocabulary and language patterns, would improve performance. Evaluations performed in Carlberger (1997) and Darragh et. Al. (1990) showed that the inclusion of an adaptive language model does indeed increase both user-acceptance and text-production rates.

2.3 User requirements

In a word, User Requirements are all about *choice*. The following is a list of just a few of the features necessary to accommodate the very varied needs of users of Eye-Control Technology. Some of the features were first identified in the COGAIN deliverable D3.1 User Requirements Report (Donegan et al., 2005), and several new required features have since been added. The widest possible variety of on-screen software interfaces needs to be offered, to expand the range of choices and the likelihood of matching the system to the user's needs and abilities.

Resizable cells and grids. Regardless of which interface software is used to allow users to interact with their eye-control system, it needs to be as flexible as possible. In a grid-based system, the grids need to be resizable, for example. One application, *The Grid*, enables the user to set cell and grid size as needed to optimise performance, instead of requiring the number of cells and grid size to be fixed, as with some dedicated eye-control systems (Fig. 3).



This is the way that the Grid cell sizes can be modified depending on the individual user's needs and ability...

This the way that you can type with a 4 x 4

about
able
above
aberrations
abundantly
ability
qu w e r t y u i o p ←
shift a s d e g h j k l ?
space z x c v b n m . !"#@
_s _es _est _iest _ed _er _ier _ent _ing _ly speak
and are can for from had have how that the clear
then there this was what when which will with you home
a b c d e f g h i
j k l m n o p q r
s t u v w x y z
EDIT

Figure 3. Both of these grids were designed with the same framework programme. The programme is flexible enough to be used by those who can eye-point accurately (left) as well as those who cannot eye-point accurately and need larger cells (right).

A range of input methods. Users require eye-control software that can accommodate a wide range of input methods in *addition* to eye control, whether (a) to change from one method to another if and when their condition changes, (b) to distribute the physical load as necessary, (c) to enable multimodal access, or (d) to use their technology in a wide range of environments, as some eye-control systems do not work well out of doors. There are many reasons for accommodating multimodal access. For example, while many users would rather stick to dwell-select or blink as their mouse-button equivalent, others would prefer to use access methods in combination, such as eye control and voice, eye control and switch, and so on.

One example of a user requiring choice of access method might arise if eye movement itself becomes difficult or impossible as the user's condition progresses. This very concern was shared by Danish ALS patient Arne Lykke Larsen, in an e-mail sent to COGAIN system developers in 2005:

> *'Many final-stage people with ALS (PALS) experience that their eye muscles become weaker and weaker. This is contrary to what it says in most textbooks on ALS, but it is actually happening. I can mention that some Japanese PALS, who have survived 20 years using a ventilator, can only stare straight out. Secondly, many final-stage PALS take drugs (scopoderm, atropin, etc.) against salivating. Unfortunately, these drugs interfere with the sight, leading to loss of precision and accuracy.'*

Supplementing gaze selection (i.e. pointing) can take various forms, including the utilisation of a user's weak bio-potentials—EOG, EMG, or EEG—which can be picked up from skin on the user's head. Bio-potentials can be used to drive single switches or position an analogue pointer. Single-switch selection of GazeTalk is made possible by an auto-scanning feature. Dasher supports one-dimensional pointing by analogue signals.

Choice of symbols vs. text input. Much of the software that is written for eye-gaze systems is restricted to eye writing using text only. However, from COGAIN's experience with users so far, it is clear that they require *much more* from their eye-control applications. They require, for example, a choice of symbols vs. text (see Fig. 4), a choice of text styles and colours, a range of speech facilities, and a choice of languages. Many people with disabilities communicate using symbols instead of text. For some, this is necessary because they do not have a high level of literacy. For others, this is through choice, as they regard it as a quicker and more efficient form of communication. Indeed, many users of symbol-based systems (e.g. *Minspeak*) are able to communicate more quickly and efficiently than those who use text-based communication. For this reason, users need to be able to choose symbols or text in their eye-control applications, whether for writing, social communication, or both.

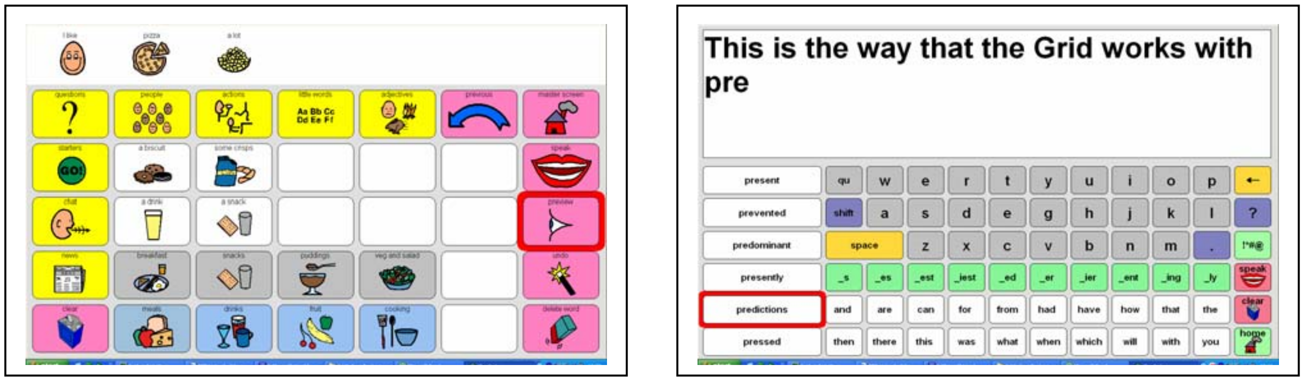

This is the way that the Grid works with pre

Figure 4. With a powerful framework programme, users can choose whether to use symbols or text.

A selection of text styles and colours. Literate users may find it preferable or even essential for their text output to be presented in a specific way. For this reason, there needs to be a full range of text output styles, including choice of font, font size, and foreground and background colour (Fig. 5).

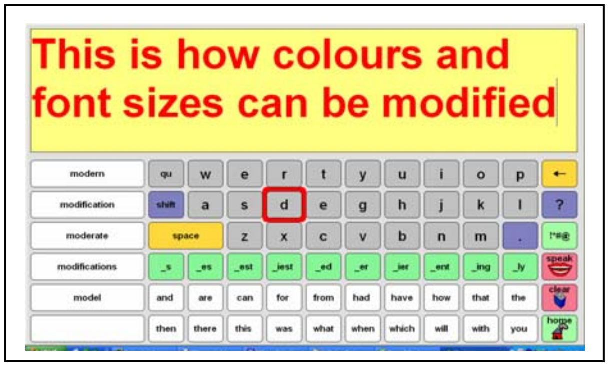
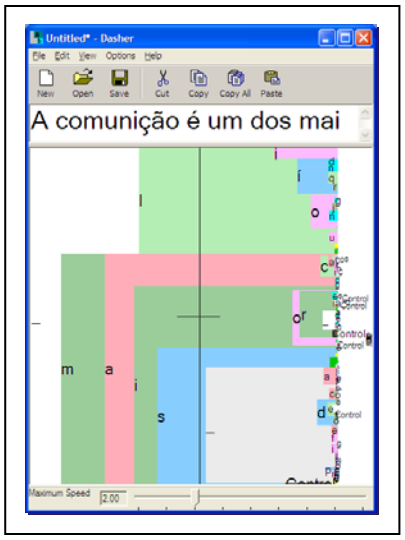

This is how colours and font sizes can be modified

Figure 5. Many users with physical disabilities have a visual impairment of some kind.
With a powerful framework programme, a wide range of fonts, background colours,
and more are available to meet individual needs.

A range of speech facilities. When using eye control for writing or generating symbols, the user needs to have the option to receive auditory feedback, with the computer speaking each letter, word, or phrase. This would reduce the need to visually double-check what is being written (Majaranta et al., 2003), or even eliminate that need altogether. Eliminating the need for this visual check would 1) speed up the writing process, and 2) overcome any difficulties related to the so-called ‘Midas Touch’ – the unintentional activation of a function during a visual search. When using eye control for social communication, a range of speech facilities would, of course, be essential. For example, users should be able to set the speed of spoken output, as well as the gender and age of the synthesised voice, according to personal preference.



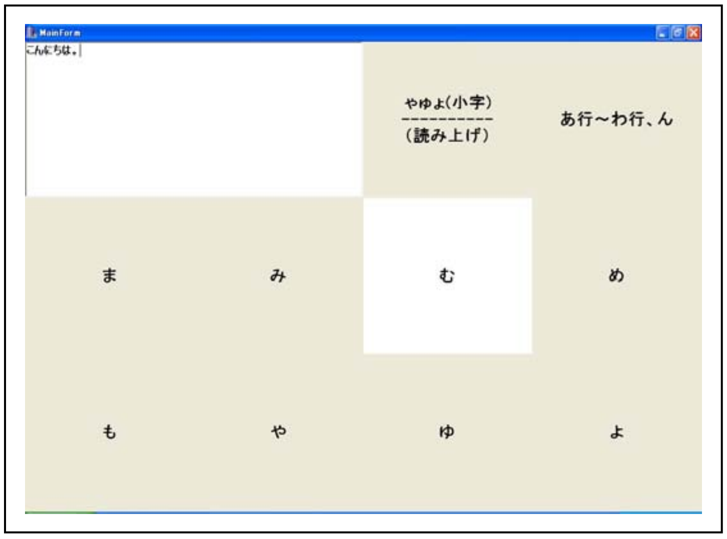
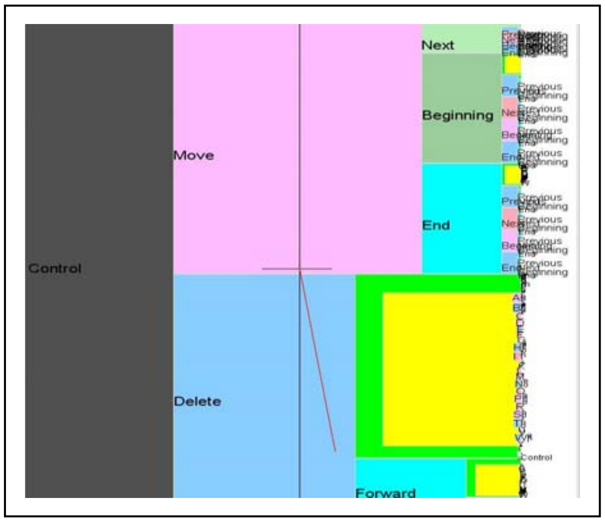


Figure 6. Portuguese is one of more than 100 languages available for use with Dasher (left).
GazeTalk comes in Danish, English, and Japanese versions (shown at right).

A choice of language. Clearly, eye-control software should support as many users, globally, as possible. Therefore, the option to use as many languages as possible should be provided. Dasher, for example, is available for use with more than 80 languages (see Fig. 6, left).

Editing facilities. When writing text for storage (and not just type-to-talk), composition and visual presentation (layout) become important factors to consider. A range of text-editing features should be provided. This is particularly important, for example, for users who are in the workplace. Figure 7 shows how basic editing functions may be activated in Dasher (left) and GazeTalk (right).



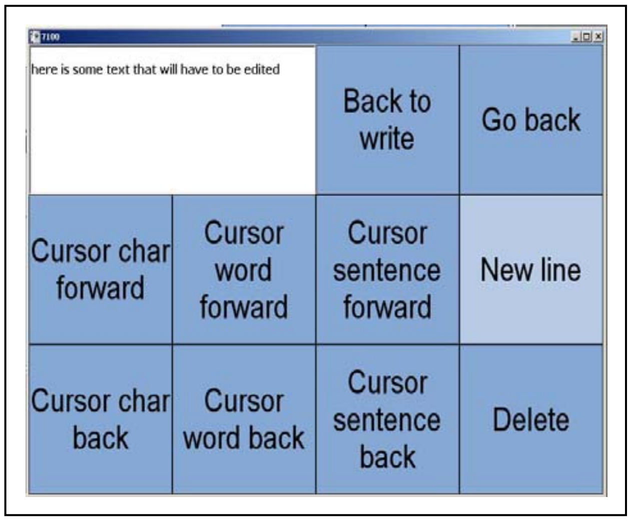


Figure 7. Basic editing functions in Dasher (left) and GazeTalk (right).

Accommodation of poor spelling. As previously mentioned, many typing systems use a language model to speed up typing. Some of the language models, however, are unable to predict what the user is typing if the user is not spelling the word correctly. Consequently, the word predictor will not provide any full-word suggestions. In cases like this, the user should be able to keep typing in his or her own (possibly incorrect) way, being supported by, for example, an alphabetic ordering of character keys. This is the case for both Dasher and GazeTalk.

The ability to handle national language and special characters. Most standard keyboards handle non-standard-letter localisation in an unsatisfactory or inefficient fashion. For instance, QWERTY relegates the Danish letters 'æ', 'ø', and 'å' to the undesirable far-right position (right-hand little finger), and the multi-tap input method used on many mobile phones requires as many as eight and a minimum of four key presses to enter these characters, depending on implementation. For almost all keyboards and input methods, the character '@' requires complicated combinations of keys that are otherwise rarely used, even though the '@' is essential for writing e-mail addresses. This issue is especially relevant for Japanese writing, in which Hiragana, Katagana, and Kanji characters all should be easily available.

Easy shift between national language and English version. Say, a German user would like to send an e-mail to an English friend. The German would appreciate being able to shift to an English language model without having to change to another writing system. In other examples, support for bilingual writing could be a necessity.

Adjustable speed. Novice users often need more time to master a writing system than trained users do, and at the end of the day, even trained users might prefer a longer response-time window than they use when they are not tired. This is relevant to users of the dwell-select feature, for example. Beginners with this method might use a 1.0-second dwell select. However, with practice and flexible software, some can achieve dwell-select speeds as fast as 0.2 seconds.

Access to system commands from within the eye-writing system. Users often need to switch to operating applications other than the writing system or to execute basic system commands. Therefore, it should be possible at least to minimise the eye-writing system with buttons big enough for eye control (as can be done with GazeTalk) to make room for other applications, or to execute commands from within the program itself. For instance, there are plans to integrate Dasher with SAW (ACE Centre) to enable the user to access and control applications from within Dasher itself.

3 Architectures of eye-writing systems

Chapter 3 presents the results of extensive discussion between Work Package 4 members at a research retreat held in Copenhagen, May/June 2005. The agreement presented in the executive summary has been circulated among all Work Package 4 members following the retreat.

The purpose of investigating the architecture of eye-typing systems is threefold: 1) to remove obstacles to innovation and implementation, 2) to improve the usability of eye-typing systems, both for users ('ordinary' usability, i.e. ergonomics and functionality) and system managers (installation, configuration, maintainability, etc.), and 3) to identify likely areas for further scientific investigation.

Our overarching strategy is a conventional one: analyse the systems, identify points of commonality for possible standardisation, and investigate them. Further, we identify crosscutting issues, which impact most or all identified components.

It is axiomatic in both science and engineering that any solution can be made arbitrarily complex. On the other hand, developing a complex set of standards takes time, and is not always warranted. Hence, in the interest of providing workable guidelines quickly, we have chosen to analyse the items in each group and categorise them as 'short-term recommendations', 'long-term recommendations', and 'merits investigation'.

The intention is that most, perhaps all, short-term recommendations—which all should be easily implemented and useful in terms of interoperability and the reusability of components and data—be agreed on and implemented by all participating development teams.

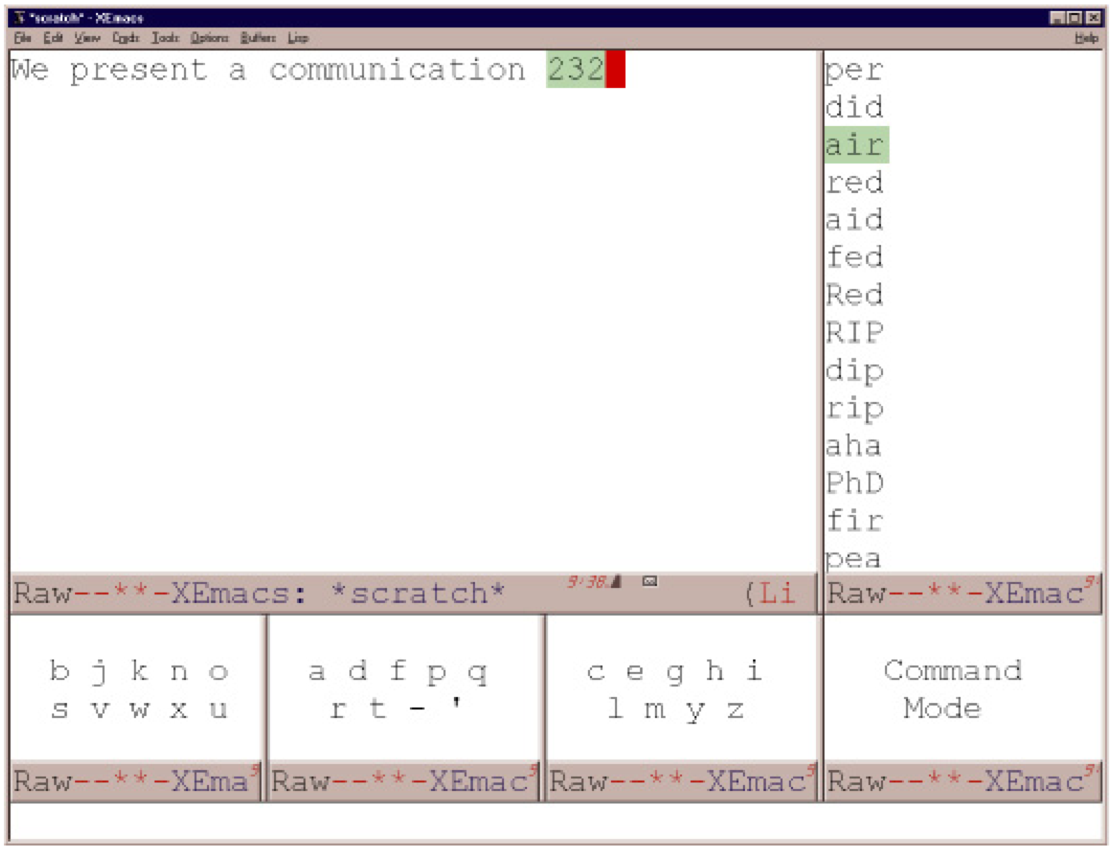
To ground the analysis in the realm of current eye-writing systems, the architectures of three COGAIN systems will first be outlined.

3.1 Overview of the UKO system and its external links

The communication aid UKO-II (Kühn and Garbe, 2001) resides on the programmable and extendable XEmacs text editor. XEmacs provides many text-entry and manipulation functions useful in the text-entry context. Operating system support, basic applications such as e-mail, and a development environment including extensive documentation are at the programmer's fingertips. All components of the communication aid dealing with input or output have been implemented as Emacs Lisp modules.

UKO-II provides a keyboard with a four-button keyboard (see Fig. 8). The upper-left panel represents the text-editing buffer. The upper-right panel lists all word suggestions for the ambiguously typed word currently under consideration. In the lower part, all keys are shown. The number of buttons has been specified in advance. This parameter depends on the user's motor functions or on the buttons available on the device. If fewer than four physical buttons are available, the keys have to be selected on a virtual keyboard via scanning. A genetic algorithm was used to calculate a near-optimal distribution of letters in order to minimise the length of suggested word lists. The keyboard of UKO-II is tailored to a user with cerebral palsy. No more than four buttons can be accessed directly. Three buttons provide ambiguous-letter keys labelled by sets of letters for typing in the text-editing window. The fourth button invokes meta-level commands such as letter deletion and word disambiguation.

Words are entered by pressing the corresponding ambiguous key once for each letter. Once the word is completed, the user disambiguates the input by selecting the intended word in a list of hits provided by the language model. Figure 9 depicts the situation after the word ‘aid’ has been typed—by pressing the second, the third, and the second button again (key sequence <2> <3> <2>)—and before the user selects the targeted word in the list of suggestions. If the target word is not known to the system, it is possible to spell the word and include it in the lexicon for future use. Other actions in the command mode provide text navigation and editing as well as invocation of the speech-output system. These actions are triggered either by overloading the three letter keys with commands, or by entering and disambiguating a command name.



```
We present a communication 232
```

```
per
did
air
red
aid
fed
Red
RIP
dip
rip
aha
PhD
fir
pea
```

b j k n o s v w x u	a d f p q r t - '	c e g h i l m y z	Command Mode

Figure 8. UKO-II after the word ‘aid’ has been typed by pressing the second, the third, and the second button again (key sequence <2> <3> <2>).

The input for selecting keys in UKO-II can be any mouse or key event input device, including switches, head or eye tracker. If only one or two input signals are available, UKO-II contains integrated scanning modes for automatic, sequential and groupwise scanning. The code is extended right now to facilitate also the input of regular Emacs commands via UKO-II. In an ongoing project, UKO-II is connected to a robust eye tracker being developed at the University of Koblenz-Landau that detects four or more specific eye end positions.

On the output side of UKO-II, a speech synthesizer and a GIDEI interface has been integrated. The speech synthesizer interface provides access to the Microsoft Speech API via sockets with the help of an auxiliary socket server and thus allows speech output via any speech synthesizer following the SAPI. Another interface that can be used within UKO-II is the GIDEI protocol that allows transferring key and mouse commands via the serial ports to another Windows computer. With GIDEI, it is therefore possible to control an external computer using the preferably mobile computer running UKO-II.

3.2 Overview of GazeTalk and its external links

The goal of the GazeTalk (Hansen et al., 2001) project was to develop an AAC system that is based on eye tracking, that supports several languages, that facilitates fast text entry, and that is not only sufficiently feature-complete to be deployed as the primary AAC tool for users, but also sufficiently flexible and technically advanced to be used for research purposes. The system was designed for multiple target languages, initially Danish, English, and Japanese, and support is currently being extended to Swedish and Norwegian. The system has a built-in voice-output subsystem that supports the use of either external (usually synthetic) voice using the SAPI, API, or the Windows clipboard, or an internally developed digitised voice system. This digitised voice is currently available in Danish only, with a vocabulary of approximately 33,000 words.

The motivations for the project were to explore and eventually deploy eye tracking as an input modality for ALS patients, and to explore issues in text input in constrained user interfaces (UI), such as mobile devices. We explore the relationship between text input in AAC systems and mobile devices extensively in (Johansen and Hansen, 2005).

The GazeTalk system consists of three elements:

- The actual GazeTalk programme, which implements the basic UI elements and functionality (e-mail, voice output, saving and loading documents, etc.).
- StateEditor, a layout editor used for designing and building the layouts used in GazeTalk.
- A library of layouts built with StateEditor, designed for various situations (deployment, research, etc.).

All layouts in the library use the four-by-three matrix format shown in Figure 6 and 7, which allows for a maximum of twelve active on-screen buttons.

GazeTalk and StateEditor support various text-input modes, which include direct selection of letters, letter and word prediction/completion, and several types of ambiguous or clustered keyboards. The current standard configuration is the ‘one-direct’ layout. This layout has been tested by simulation and experimentation, and was shown in theory to be slightly more effective than using four suggestions selected using a two-step procedure. In actual use, this layout exhibited similar performance, less user confusion, and greater increase in text-production speed than did the previous standard layout incorporating eight suggestions with two-step selection.

GazeTalk contains a wide variety of reporting functions that allow an experimenter to measure variables such as average selection speed, words per minute, pointer trails in terms of entry, exit, and time spent per button, the type and specific content of buttons inspected and selected, as well as keeping track of the user’s navigation within the system.

A core feature of the system is the use of statistical language modelling to increase the input rate and allow for a relatively low number of on-screen targets (currently ten active targets/buttons, with two of the available twelve positions reserved for text display). GazeTalk employ both letter prediction and whole-word prediction. Both prediction subsystems (letter and word) are implemented using a relatively simple Markov model, as we are only concerned with retrieving the ranks of words. The word-level model uses a 3-gram back-off model (Katz, 1987). Originally, the language model was trained using approximately 800,000 words of Danish text gathered from newsgroups and e-mail in-boxes.

We have since implemented a better-performing language model. As reported in Johansen et al. (2003), it was decided to do so initially through increasing the size of the vocabulary and training corpus. The open-source Danish dictionary *Den Store Danske Ordliste* was used as a basis for cleaning and organising a corpus collected from all available Danish Usenet groups. That resulted in a final corpus consisting of approximately

35,000,000 words, with a vocabulary of approximately 89,000 words, elaborated using the same methodology and used when constructing the initial language model (described in detail in Johansen et al., 2003). This new language model used the same algorithms as the original model, reimplemented for improved space efficiency in the face of a training corpus that is nearly two orders of magnitude larger than the original corpus.

GazeTalk is implemented for the Microsoft Windows platform using the Borland C++ Builder (BCB) development environment, which is largely compatible with the better-known Delphi Pascal–based system, also from Borland. Most features are implemented using the BCB/Delphi-specific ‘VCL’ system for reusable components, but almost all non-interactive functionality is implemented using standard C++ classes and facilities. This specifically includes the language model implementation, whereas the dwell-sensitive buttons and various text-input fields are implemented in a very platform-specific way, which is probably not immediately reusable in other development environments, even those based on Windows and C++.

The internal component structure largely follows the component breakdown developed by the ‘Camp COGAIN’ workshop (Fig. 9), except that no attempt was made to consolidate API and functionality for input or output devices.

3.3 Overview of the Dasher system and its external links

Dasher (Ward and MacKay, 2002) is implemented as a stand-alone application in a modular fashion. The primary separation of functionality is into a 'core', which is designed to facilitate cross-platform development, and a user interface, which is implemented on a per-platform basis. All modules are implemented in C++. The core handles the display of the main Dasher canvas, as well as the dynamics, language model and so on. The user interface embeds the core, as well as providing a simple text editor, file handling facilities and connectivity with other components of the host operating system.

The primary development platforms for Dasher are Microsoft Windows, Windows CE and Linux through the GNOME desktop environment. The core is presented as a custom GTK+ user-interface component under Linux and as a C++ class designed to facilitate embedding under Windows and Windows CE. Implementations for Apple OSX and Linux running QT and GPE also exist.

The core is subdivided into a number of modules representing distinct functions, which are designed to be 'pluggable', ie to minimise the overhead in replacing them with alternative versions. The key components are the dasher model, which handles the dynamics of the display, the 'view', which actually renders the model and the language model, which makes predictions used to update the model. The basic language model implementation uses a variant of Prediction by Partial Match (PPM) to make predictions. Other models currently under development use a more sophisticated word-based model and enable Japanese to be entered as Hiragana and then converted to Kanji.

All text functions of Dasher make use of Unicode, which greatly facilitates internationalisation. The language in which text is to be entered may be selected by the user, which decides the alphabet which is presented as well as customising the predictive model using a training text for the appropriate language. Alphabets are specified using an XML file format, and over 100 such files are currently available. The language model is adaptive, and can therefore learn specific vocabulary, and the user can import their own text (for example past e-mail) to further improve the predictions. The colour scheme is configurable in a similar way to allow for users' preferences.

As well as simple text entry, Dasher implements a 'control mode', which allows the user to control Dasher, perform basic editing tasks and access additional functions such as speech through the Dasher interface itself rather than relying on external user-interface components.

Input to Dasher is flexible and can be through a pointing device (e.g. an eye-tracker or mouse), a device capable of providing one or more continuous signals (for example a breath interface), or any number of discrete buttons. Various configurations exist to modify the behaviour of Dasher so that it is more suitable for particular input devices, for example, a one dimensional mode allows both speed and direction to be controlled by a single input signal, an eye-tracker mode implements automatic calibration and modified dynamics and button modes implement various methods for driving Dasher through a button interface. As with the rest of Dasher, the design is such that new devices can be interfaced with relative ease.

In addition to editing functions, the user interface provides speech synthesis through the GNOME speech libraries on Linux and the Microsoft Speech API on Windows, and control of other applications through the platforms' respective accessibility APIs.

3.4 Identifying the components

As part of the Ph.D. course on Under the Hood of Advanced Writing Systems (held during the 'COGAIN Camp' workshop in Copenhagen, May 30–June 3, 2005), the participants analysed and identified the necessary components in eye-typing systems. These components are graphically represented in Figure 9:

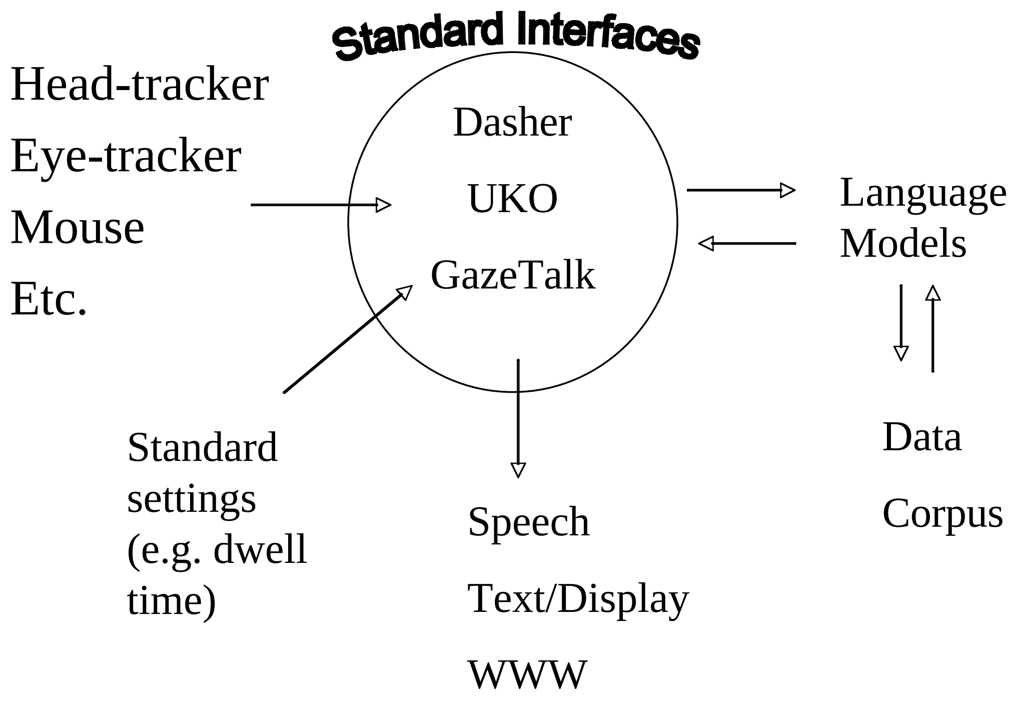


Figure 9. Graphical representation of components essential to eye-typing systems.

The major groupings of components are thereby:

- Input devices
- Output devices
- User profiles and other settings
- Language model implementations
- Language model data (corpora)

3.4.1 Input devices

One might expect that the majority of input devices, if not all, could be categorised by a few simple properties and traits, such as dimensionality (1D, 2D, 3D) and number of buttons (e.g. for mice). Unfortunately, even a casual investigation of input devices reveals that this is a surprisingly complex area thanks to the wide variety of input devices available. A non-exhaustive list of devices already supported by current COGAIN applications includes eye trackers, buttons, five-way navigation (digital joystick), breath control, keyboard, mouse, tablets, pens, and touch screens.

First and foremost, communication with input devices must often be two-way, in order for the application to be able to query the device on various parameters, and for the device to communicate its current status. In the case of some device classes, specifically gaze tracking, it is often desirable to allow for a very close integration with the input device indeed, as the interaction between the application and the gaze tracker may be very complex.

Possible properties for input devices:

- Accuracy, which may change dynamically
- Input rate
- Dimensionality
- Relative or absolute positioning (e.g. mouse or touch tablet)
- Discrete or analogue (e.g. five-way navigator or analogue joystick)

Furthermore, some devices may allow for calibration, or other features that are neither suitable nor necessary for all types of devices.

Short-term recommendation: As the variety of input devices makes attempts at categorisation exceedingly difficult, we recommend the initial focus be to define suitable interfaces for the most common input devices, mainly gaze (obviously), mouse, five-way navigators (joystick, arrow keys, etc.), and single-switch and direct pointing devices (digitisers, touch-screen, pen, etc.).

Long-term recommendation: We recommend that the work on standardising the means for interaction with input devices be considered an ongoing process, with the dual purpose of improving existing interfaces and extending support to other input devices.

Merits investigation: As topics for long-term investigation, we recommend additional work on identifying points of communality and an investigation of ways to ‘disguise’ one device as another (e.g. use a gaze tracker as a mouse). This is interesting both from a practical, usage, or reuse-oriented point of view, and also from the scientific point of view (see e.g. Bates and Istance, 2003).

3.4.2 Output devices

Because the primary focus of COGAIN is on text input and output, it is a relatively simple task to define and implement the short-term goals for this module. Our immediate target output devices—currently text-to-speech, Braille, and printers—all accept a text stream as their primary input. More advanced use of output devices, or a more-inclusive definition of ‘output’, can thus be considered a long-term task. In order to facilitate interoperability and reuse on several computing platforms, the recommendation of the workshop is that the UTF-8 format be used, as it encodes all national characters in a standardised, unique way.

Short-term recommendation: Design a simple interface for output that accepts a UTF-8 stream of text. Implement adaptors for the common output devices that conform to the interface. Some facilities for feedback may be deemed necessary, for example, to highlight words as they are read by a Text-To-Speech (TTS) system.

Long-term recommendation: Investigate the case for a more advanced output interface definition. Consider whether remote control of other applications (e.g. common e-mail programs, web browsers, word processors, etc.) should be considered output, and if so, which functionality should be accessible and how.

Merits investigation: This is a practical engineering task but it may also be interesting to study e.g. how an eye mouse or a head mouse performs in the role of the more conventional hand-operated mouse.

3.4.3 User profiles and other settings

This is potentially the most complex component, as a fully realised and generic implementation will represent the complexity of all subsystems, and additionally add some of its own.

The technical goal of a unified, standardised settings system is twofold: 1) to provide a standardised way for programmers to store and retrieve various settings, and 2) to provide a list of standard settings that COGAIN systems should understand and comply with. The user benefit is that it is then possible (ideally) to change from one system to another without having to reconfigure anything. For the system integrator, there is a potential labour and training savings in only having to learn one tool and understand one standard for configuration settings.

Unfortunately, the variety of the COGAIN products and their subsystems to be configured conspire against the design of a simple, well-defined, and yet inclusive system for settings. The potential complexity of input devices has been mentioned. Further, several settings will be inherent to individual COGAIN programs, or even in some cases a single mode or subsystem of one program.

In other words, an extremely detailed standard for settings will increasingly reflect implementation details for individual programs.

Further, the goal of an advanced standard for settings must balance the work necessary to design and implement it against the time and effort it would save. One should therefore consider that users today in fact rarely switch between AAC systems, and that installation and configuration are events that occur only a few times in any usage scenario. Hence, perhaps this effort is better spent on implementing self-calibrating programmes that automatically adjust to usage, improve the ergonomics of the configuration process, or both.

Short-term recommendation: Define and implement a simple API for storing and retrieving settings, preferably using XML for cross-platform support. Define a simple set of settings common to most or all COGAIN programs as a starting point. Define a way for program-specific settings not covered by these general settings to be marked, so other programs will be able to avoid trying to interpret them accidentally.

Long-term recommendation: Investigate a method for allowing off-site persistence of settings for backup and sharing. Evaluate and extend the standard settings defined previously.

Merits investigation: The above analysis might be too pessimistic or simplistic, and the actual potential ergonomic improvements and timesavings should be investigated. We advise quantifying the potential gains and required efforts, in order to determine whether an advanced system would be a case of 'You're not going to need it': spending too much time up-front, building a flexible, but byzantine and unwieldy, structure, or 'too simple': developing a solution that is too simplified a model of the actual problem—versus investing some more time in getting it 'just right'.

3.4.4 Language-model implementations

Technically, the use of a language model is optional in AAC systems. However, all of the current COGAIN systems integrate language models in the core design to such a degree that they could not possibly do without one. We expect this trend to continue in future designs.

There are several ways to implement language models. Current COGAIN systems feature:

- Context-dependant vs. static (GazeTalk and Dasher vs. UKO II)
- Word level vs. character level (UKO II and GazeTalk vs. Dasher)
- Ambiguous vs. literal interpretation of input (UKO II, GazeTalk in Tx configuration vs. Dasher and GazeTalk in normal configuration)
- Dynamic adaptation to user vs. static model (all systems, depending on configuration)

Short-term recommendation: As we have several working implementations, the main challenge is to make one or more of them available as a reusable component (see 'Licensing issues', below) and define a suitable API for interacting with it, taking platform issues into account. Initiate work on making all implementations Unicode (UTF-8)–compliant.

Long-term recommendation: Unify the number of models supported by COGAIN. It is reasonable to expect that all features could be implemented in two models (word-based and letter-based), thus reducing the amount of time spent maintaining, extending, and improving models.

Merits investigation: A possible long-term goal is the development of advanced, higher-performance models. The potential payoff is unknown but expected to be limited, as much work has been done in this area over the last three decades, with precious little to show for it in terms of increased prediction quality. Time may be better spent on general improvements of ergonomics.

3.4.5 Language-model data (corpora)

Most language-model implementations depend on training data, or **corpora,** to avoid depending entirely on text produced by the user. The base-level performance of any language model is highly dependent on the suitability of topic domain, dictionary, and size of the training data.

Some corpora are obviously already in existence in COGAIN projects, as are tools for collecting and editing corpora, distilling dictionaries, and (compiled) language model data. Still, not all major European Union languages (not to mention minor EU languages) are currently covered by the existing corpora.

Another issue is how to store and process language data collected from the user during actual use. These data are generally of great value in terms of improving predictor performance, as they are highly domain-specific.

Short-term recommendation: Store user text history as a UTF-8 text stream without additional formatting or tags. Make available corpora and tools used in the current projects, given that licensing issues can be solved beforehand.

Long-term recommendation: Design or choose standard model formats. Develop a library of corpora and (compiled) models for all EU languages. Develop a mechanism for off-site persistence of user-produced text (backup and interoperability). Develop tools for generating user-oriented corpora from previous text generated by the user (old documents, web pages, blogs, e-mail, newsgroup entries, etc.).

Merits investigation: Is anything beyond naïve storage of user text production necessary? In other words, should data be 'tagged' with task-related information, intended recipient, time of day/month? Traditional dictionary and corpora collection are not scientifically interesting tasks, as they are largely resolved and thoroughly investigated topics in the field of (computational) linguistics.

3.5 Identifying cross-cutting issues

Some issues have a potential bearing on all, or nearly all, of the aforementioned components. We identified the following crosscutting issues:

- **Cross-platform.** The existing COGAIN systems run on a variety of computing platforms, which may give rise to additional complexity.
- **Licensing.** Some COGAIN systems use an open license. Some are closed. The commercial partners may wish to further redevelop some currently open components as closed-source.
- **Privacy and security.** If the systems store extensive information on usage (e.g. text input by the user), it may be desirable or necessary to employ privacy-protection measures.

We discuss these issues further below.

3.5.1 Platform issues

As can be seen from sections 3.1 and 3.2 the applications developed by COGAIN members are deployed, and often depend, on a wide variety of hardware and software platforms. A set of recommendations must therefore be cross-platform to be relevant for current COGAIN activities.

At this point, however, we defer design recommendations for how to achieve this, as it seems more natural to agree on what should be standardised before we determine how it should be implemented. We do encourage members to consider platform issues when reviewing the items in this section, and to investigate or suggest ways to facilitate cross-platform reuse of components and data.

3.5.2 Licensing issues

A licensing issue is a legal challenge, rather than a scientific or engineering one. Nevertheless, it must be resolved before any reuse, or even sharing, of actual source code can be initiated, as the consequences of not doing so are currently unknown. What we as engineers and scientists *can* do at this point is 1) compile a list of the licenses currently in use in COGAIN projects, and 2) elect or hire a competent agency to resolve any licensing issues for us, or at least provide us with appropriate guidelines.

3.5.3 Privacy and security

One could argue that privacy and security really are matters to be resolved by the computing environment, but considering that our users are unable to log in and out in an ordinary fashion, they are also unable to utilise the protection offered by the current systems. Hence, it is both a question of (mental) ergonomics, that is, whether the user can rest assured that his or her data and actions are indeed private, which in some cases can be of legal import. In some countries, the developers of an AAC system may be liable if a user's data are shared or processed without express permission. For the time being, we propose that this be dealt with by leaving the decision up to the user, or possibly having him or her sign a waiver before use of the system is allowed.

4 Conclusions

The purpose of this task is mainly to support innovation in the niche by helping innovators avoid ‘reinventing the wheel’ in terms of analysis and implementation of common, necessary components. The secondary purpose is to provide a taxonomical framework that can guide further scientific investigation of this area.

One of COGAIN’s primary, practical, implementation-oriented tasks is to develop guidelines and reference implementations for eye-typing systems and components of such systems (preferably reusable). At this stage, we have initiated the process by performing initial analysis, which identifies the components of the systems, as well as initial work on a taxonomy for further subdivision. Partial recommendations have been completed for some components, and topics for further investigation are largely identified and categorised.

References

Ashmore, M., Duchowski, A.T. and Shoemaker, G. (2005) Efficient Eye Pointing with a Fisheye Lens. *Proceedings of GI 2005*, May 9–11, 2005, Victoria, BC, Canada. Canadian Human-Computer Communications Society (CHCCS)/ACM.

Bates, R. and Istance, H.O. (2002) Zooming interfaces! Enhancing the performance of eye controlled pointing devices. *Proceedings of the Fifth International ACM SIGCAPH Conference on Assistive Technologies (ASSETS 2002)*.ACM Press, pp.119–126.

Bates, R. and Istance, H.O. (2003) Why are eye mice unpopular? A detailed comparison of head and eye controlled assistive technology pointing devices. *Universal Access in the Information Society 2(3)*, pp. 280-290.

Bates, R., Istance, H., and Spakov, O. (2005) *D2.2 Requirements for the Common Format of Eye Movement Data.* Communication by Gaze Interaction (COGAIN), IST-2003-511598: Deliverable 2.2. Available at http://www.cogain.org/results/reports/COGAIN-D2.2.pdf

Carlberger, J (1997) Design and Implementation of a Probabilistic Word Prediction Program ; *NADA report TRITA-NA-E9751*, Swedish Royal Institute of Technology, dept. of Numerical Analysis and Computer Science (NADA), 1997.

Darragh, J.J., Witten, I.H. and James, M.J. (1990) The Reactive Keyboard: A Predictive Typing Aid. *IEEE Computer* 23(11), pp.41–49.

Donegan, M., Oosthuizen, L., Bates, R., Daunys, G., Hansen, J.P., Joos, M., Majaranta, P., and Signorile, I. (2005) *D3.1 User requirements report with observations of difficulties users are experiencing.* Communication by Gaze Interaction (COGAIN), IST-2003-511598: Deliverable 3.1. Available at http://www.cogain.org/results/reports/COGAIN-D3.1.pdf

Hansen, J.P., Hansen D.W., and Johansen, A.S. (2001) Bringing Gaze-based Interaction Back to Basics. *Proceedings of Universal Access in Human-Computer Interaction (UAHCI 2001)*, New Orleans, Louisiana, pp. 325–328.

Hansen, J. P., Johansen, A. S., Hansen, D. W., Itoh, K. and Mashino, S. (2003). Command Without a Click: Dwell Time Typing by Mouse and Gaze Selections. *Human-Computer Interaction – INTERACT´03.* M. Rauterberg et al. (Eds.) IOS Press, pp.121–128.

Johansen, A.S. and Hansen, J.P. (2005, to appear) Augmentative and alternative communication: The future of text on the move. Paper accepted for publication in The International Journal "*Universal Access in the Information Society*".

Johansen, A.S., Hansen, J.P., Hansen, D.W., Itoh, K., and Mashino, S. (2003) Language technology in a predictive, restricted on-screen keyboard with ambiguous layout for severely disabled people. *EACL 2003 Workshop on Language Modeling for Text Entry Methods*, April 13, 2003, Budapest, Hungary.

Jordan, P. (2000) *An Introduction to Usability.* Taylor & Francis.

Katz, A. (1987) Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing.* VOL ASSP-35, no. 3, March 1987.

Kurtenbach, G. and Buxton, W. (1994) User learning and performance with marking menus. *Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems.* Vol. 2, ACM, 218.

Kühn, M., and Garbe, J. (2001) Predicctive and Higly Ambigiuous Typing for a Severely Speech and Motion Impared User. *Proceedings of Universal Access in Human-Computer Interaction (UAHCI 2001)*, New Orleans, Louisiana, p. 933–937.

Majaranta, P. and Räihä, K. J. (2002) Twenty Years of Eye Typing: Systems and Design Issues, *Proceedings of the Symposium on ETRA 2002: Eye Tracking Research & Applications Symposium 2002*, New Orleans, pp.15–22.

Majaranta, P., MacKenzie, I. S., Aula, A., & Räihä, K.-J. (2003) Auditory and visual feedback during eye typing. *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI 2003)*. New York: ACM, pp. 766-767.

Rosenfeld, R. (1996) A maximum entropy approach to adaptive statistical language modelling. *Computer, Speech and Language* 10: pp.187–288.

Shannon, C.E. (1951) Prediction and entropy of printed English. *Bell Systems Technical Journal 30*:pp.50–64.

Tilbourg, H. (1988) *An Introduction to Cryptology*. Kluwer Academic Publishers.

Ward, D.J. (2001) *Adaptive computer interfaces*. Ph.D. thesis, Inference Group, Cavendish Laboratory, University of Cambridge, November 2001, http://www.inference.phy.cam.ac.uk/djw30/papers/thesis.html (verified August 2005).

Ward, D.J. and MacKay, D.J.C. (2002) Fast hands-free writing by gaze direction. *Nature 418(6900)*: 838.