Data Structures in the Design of Interfaces

G. Marsden[1], H. Thimbleby[2], M. Jones[2] and P. Gillary[2]

[1]Department of Computing Science, University of Cape Town, South Africa; [2]School of Computing Science, Middlesex University, London, UK

Abstract: Computer science algorithms can be used to improve user interfaces. Using data structures as a source of design ideas, a new interface was constructed for a cellular telephone handset. Once implemented, a user experiment was conducted which showed that predicted improvements in usability were confirmed with real users doing realistic tasks.

Keywords: Cellular handset interaction; Menu design

1. Motivation

Around 80% of cellular telephone users only use their handsets to receive calls and make calls dialled directly on the keypad [1] – in other words, they do not use any facilities other than those provided on the most basic of land-based telephones. One might speculate that users may not want any more functionality, but the nature of a cellular telephone almost demands that the user investigate configuration options (for example, because the handset will move from noisy to quiet environments, the user will need to adjust the ringing tone to an appropriate level). It is also clear from the calls received by help-lines set up by the cellular service providers that users do want to access features of their handset, but are hindered by the handset's interface.

The work presented in this paper was initiated by a cellular service provider who was concerned not only about how much the help-lines were costing the company, but also about lost earnings because users were not bothering to access premium rate services (e.g. voice mail, traffic reports). Both problems are primarily due to overly complicated configuration options on their handsets. These same concerns are also echoed by [2], another cellular service provider who has also become concerned about the handsets they are supplying to their customers.

The original aim of the work described in this paper was to design a new interface to cellular handset features which users could access more easily than existing interface designs. The paper reports several design alternatives and the results of user testing the final design.

1.1. The problem

The interface for most cellular handsets is a hierarchical tree of menu choices. Features are grouped logically by operation and the overall structure is navigated by providing parent, child and closest-sibling navigation at each node (see Fig. 1).

Whilst menus are effective in overcoming command recall problems on a typical computer monitor, it is not clear that they are an effective solution for structuring interaction on a cellular handset, which has a greatly reduced screen size. Typical handset screens can display only one menu option at a time, forcing users to remember the other options in each menu. As the menus are nested (sometimes to a depth of four levels) it is little wonder that users become confused about their location in the menu structure and where to find the command they want. This is often compounded by the classifi-

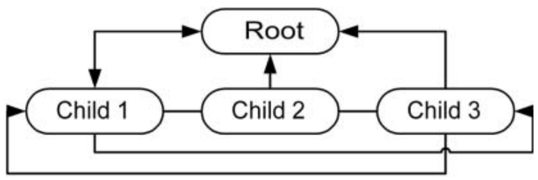


Fig. 1. Interface menu hierarchy. Scrolling right from the third child returns to the first child. Scrolling left from the first child moves to the third child.

cations and ordering used in the menus, which may not agree with how the user would choose to classify functions.

We agree with Alexander that hierarchical tree type classifications are not a natural way of organising structures for humans [3]. (Alexander's main argument is that trees are good for designing, because the designers know the classification system, but they are not good for users who do not know the classification, or perhaps even the names and relations of the commands). The user of a cellular handset must guess how the handset's designer would classify a particular function – for example, does a function to set the ringing volume belong in the 'Tones' menu or the 'Phone Settings' menu?[1]

Besides problems of menu structure awareness, displaying one menu option at a time means that the user must perform many key presses in order to navigate to the desired option. Increasing the number of key presses required will obviously increase the likelihood of making an error in input.

To solve these sorts of problems of interaction with cellular telephone handsets, it is necessary to develop a solution which was not based on traditional hierarchical menus but employed an alternate interaction paradigm.

2. Where to Look for a Solution

Although there are handsets coming on to the market which have large, touch screen displays, small screens are likely to remain dominant in the handheld market for a long time. Even if higher-resolution screens are made available, ageing populations (with poor eyesight) or usage requirements in harsh environments (bad lighting, while driving) suggest that solving small screen interaction will remain an important problem. The present work was concerned only with standard handsets with small screens and push button-style interaction. There are many comments we could make on the physical design of the handsets (such as better affordances on volume buttons) but, in the interests of brevity, we shall concern ourselves only with the software of the interface rather than the hardware. Our comments and general results (if not the absolute timings) therefore apply even if the hardware is changed radically: for instance, if it was pen based or even speech operated.

[1]On the Nokia 5110, it belongs in the 'Tones' menu.

What then can be said of the interface software? Although working at institutions which employ human-computer interaction researchers from a diversity of backgrounds, the authors of this particular paper are all computer scientists (albeit experienced in HCI). It seemed logical, therefore, to start investigating the interaction problems by considering the characteristics of the menu as a data structure. To start our investigation, we first of all conducted an analysis of an existing handset.

2.1. Existing handset

To serve as an illustration of our analysis, we present the example of a Nokia 5110 handset. The results of the analysis are in no way unique to this handset or manufacturer, but the 5110 is interesting as it is a very popular handset. Other people reading this paper and wishing to repeat our work will find the Nokia 5110 readily available. Furthermore, the Nokia 5110 is marketed on its ease of use; the following is taken from 5110 promotional material [4]:

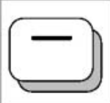
> Use your phone as you want. Send short messages, save names and numbers, select a new ringing tone – all with the press of a single key, the Nokia Navi^{TM} Key.

The Nokia handset has 74 functions which are of interest to us. The handset provides other functions that are not accessible from the menu (such as keypad locking and ear-piece volume adjust), placing them outside the scope of our study.

In this analysis we were interested in the cognitive overhead imposed on a user trying to access a menu function. There is a distinction between *accessing* a function and *activating* a function: activation requires the user to enter some data, which will be different for different functions; accessing a function is the act of locating the desired function in the menu structure without activating it. Relative costs of function accessibility can be easily calculated by counting key presses – keys being the only way to navigate the menu structure. The navigation keys are as shown in Table 1.

We shall assume a naïve user, who has not memorised the positions of every function in the memory structure and must therefore scroll to find the required function (this is likely to be the

Table 1. Keys used in navigating the current Nokia 5110 menu structure

Navi		Select the currently displayed function, or if it is a menu item, display that menu
Up		Scroll up the currently displayed menu
Down		Scroll down the currently displayed menu
Cancel		Go back to the previous selection

case for most users). Therefore, the menu structure should support both direct function access (where the user knows what function they are looking for) and browse-style accessing (where the user wishes browse through all the functions provided by the handset).

Although the menu structure was organised to favour breadth over depth, as recommended by Miller [5], the user must perform **8.2** key presses on average to access a function. To access the most nested function in the menu, a maximum of **15** key presses are required. For a browsing interaction, the user would need to perform a minimum of **110** key presses to access every function! (A distribution of key presses is given in Fig. 2.)

These calculations represent a best-case scenario, where the user makes no errors in input and is able to recognise the correct menu option when it appears on the screen. If the user makes errors, they will take longer; and, of course, if they do not recognise the function they want, it could take forever! Indeed, users may

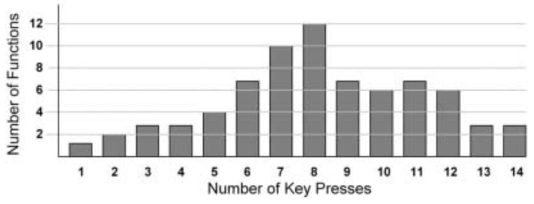


Fig. 2. Plot of the number of functions accessable from a given number of key presses.

not be sure when first they see a function that it is the actual one they want, so they would then scan the rest of the function names and eventually try to come back. Such usage is very slow. In any case, as our experiments show, optimal use is unachievable in practice – in fact, many subjects in our study became caught in loops within the menu structure, never finding the target function.

The brief analysis of the 5110 depends on other assumptions, such as the distribution of function access. For simplicity we assumed all functions were used equally often, but if an easily-accessed function were used more often, then the average would decrease. A more thorough discussion of function weighting can be found in [6].

2.2. Alternate design

A data structure that requires an average of 8.2 key presses to access a given function seems somewhat sub-optimal. Treating this as a computing science problem, one way to improve the menu tree is to restructure it as a balanced binary tree. Users searching for a menu item would navigate on the alphabetic order of the function name they were searching for. At each node in the tree, users would either select the function name at that node, or choose to navigate down the node's left or right branch. This scheme uses the four navigation buttons slightly differently, as shown in Table 2. This solution would reduce the average cost of selection from 8.2 key presses to 5.4 key presses (Table 3).

Table 2. Keys used in navigating the new menu structure

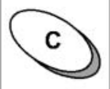
Navi		Access the currently displayed function
Up		Display a function earlier in the alphabet
Down		Display a function later in the alphabet
Cancel		Go back to the previous selection

Table 3. Method of calculating average key presses for binary tree menu

There are 74 functions and, as they have different alphabetic names, a balanced tree gives them unique position. A binary tree with six levels can accommodate 63 functions; with seven levels it can accommodate 127 functions. 74 functions therefore require six full levels and a seventh level only 15% ((74–63)/64) full. The way binary trees work, one command (e.g. the most popular) can be accessed immediately by pressing Navi; two commands can be accessed by pressing either Up or Down and then Navi; four commands can be accessed by pressing either Up or Down twice and then Navi . . . and so on. The average number of key presses required to access a function (rather than activate it) is calculated as $(1 \times 1 + 2 \times 2 + 4 \times 3 + 8 \times 4 + 16 \times 5 + 32 \times 6 + 11 \times 7)/74$, which is 5.4

Furthermore, the worst-case search path is reduced from 15 to 7 presses. However, by maintaining the hierarchical tree structure, the task of visiting every node in the structure is still daunting, requiring the user to make 148 key presses.

Of course, to remove the navigational difficulties of a tree structure, we could flatten the structure to a linear list. This would allow users to visit every function with only 74 key presses; what is more, the key being pressed would be the same every time. However, with a list, the average search time is 37.5 key presses and the worst case search requires 74 key presses.

It was clear from these preliminary design investigations that an alternative solution would need to be sought which supported both directed and browsing style access.

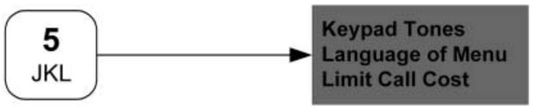
2.3. Final design solution

The final design solution was based on the technique of *hashing* [7]. Hashing usually involves calculating a memory or storage location from a key value. For cellular handsets, we developed a hashing function based around the fact that each numeric key on the keypad is also used to represent alphabetic characters: for example, the '1' key also has the letters 'abc' printed on it. Therefore, it was possible to build a hash table where the names of the functions were represented as numerical strings. For example, the word 'call' would be encoded as '2255' – the '2' key contains the letters 'a' and 'c'; the '5' key contains the letter 'l'.

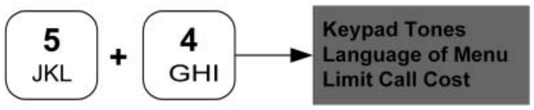
To retrieve a function, the user merely spells out the name of the target function using the numeric keys. Whilst the input may be ambiguous (each key represents between three and four distinct letters), the input can be disambiguated by the hash table in 2.7 key presses (on average).

Previously, other systems have attempted to exploit the letters on a telephone keypad. Rau and Skiena [8] give an excellent review of keyboard text entry using numeric keys, though their aim was to reconstruct English text without any interaction. They achieved 99% correct identification of characters. The T9 system [9], again intended for natural language text entry, ensures accuracy by interactively asking the user to select from ambiguous choices. More commonly, systems such as SRS from IBM [10] require the user to enter unambiguous input in the first place. This inevitably results in many more key presses; for example, to enter the letter 'c', would require the user to press the '2' key three times (once for 'a', twice for 'b' and three times for 'c').

In the situation presented here (namely, using the keypad to access menus) the user is not inputting novel data. Therefore, any input can be unambiguously mapped to one of 74 possible function names. Selection occurs as follows.



Let us imagine a user wishing to select the 'Limit Call Cost' function (a full list of functions can be found in appendix A). The user presses the '5' key (as it contains the letter 'L') and the screen presents all menu options starting with 'J', 'K' or 'L'. In this case there are three matches as shown below:



The user now presses the '4' key which, in this case, unambiguously selects the 'Limit Call Cost' function – this is the only function whose first letter is 'J', 'K' or 'L' and whose second letter is 'G' or 'H' or 'I'.

This algorithm was first prototyped in Bongo and then implemented as a full Java 1.1 applet. The Java version can be seen in Fig. 3.

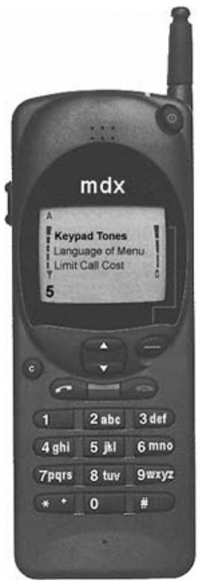


Fig. 3. Screenshot of the new design based on hashing.

3. Combining Function Access with Dialling

Most handsets are normally in 'dial' mode, when pressing any digit key starts entering a telephone number. Menu selection of functions is achieved in a separate mode, which on the Nokia 5110 is first entered by pressing the Navi key. When in 'menu' mode, the numeric keys cannot be used for entering a telephone number; indeed, the numeric keys now function as a short cut, a faster way of navigating the function menu.

Since our approach uses the numeric keys naturally to navigate the function menu, it is important to reduce this mode confusion. We achieved a modeless design as follows.

Users use the keys to enter a telephone number or to spell a function name. If the user wishes to place a call, then they simply press the call button. If they wish to select a function, they can press the selection button (in our implementation, the button to the right of the scroll keys). What may not be clear from the diagram (Fig. 3) is that the top menu item is in bold which, in conjunction with the line down the right-hand side of the screen, indicates that this function will be selected when the selection button is pressed.

As well as spelling the name, the user is free to use the scroll buttons to move the function they require into the selection position. Therefore, if the user, whose screen we see in Fig. 3, had been looking for 'Limit Call Cost' they need not spell the rest of the name but merely press the 'Down' key twice. When searching for a particular function, it is the use of the scroll key that allows us to reduce the maximum key presses from nine (using hashing only) to a maximum of six (with combined hashing and scrolling). It is also the scroll keys that allow the functions to be searched in a linear fashion, requiring 74 key presses to access every function. (It should be noted that this is an experimental prototype and issues of button placement were not considered.)

In summary, the use of hash tables provides a solution which blends the best attributes of trees and linear lists to build a structure similar to a B+Tree (similar in the fact that a node can be accessed by an indexing technique – like hashing – or accessed linearly as part of a linked list). The users of these data structures are not computers however, but ordinary – emotional and inconsistent – humans. In order to test the effectiveness of this new interface paradigm, it was necessary to conduct usability tests.

4. Experiment Design

The aim of the experiments was to test the following hypotheses:

1. The hash based interface would require fewer key presses for function access than an interface based on a traditional menu structure.
2. The users of a hash based interface would require less time to access a function than an interface based on a traditional menu structure.

4.1. Subjects

In total, 30 subjects took part in the test; they consisted of students, academics and administrative staff from a variety of university departments. Two distinct groups were required for the experiment. Subjects were rated on their experience with using cellular telephone handsets to ensure that each group consisted of equal numbers of novices (who had never used a mobile telephone) and experts (who were able to change at least one setting on their handset). An even gender and age mix was also ensured in each group.

4.2. Simulations

Two handset simulations were created for the experiment. One used the menu structure of the Nokia 5110; the other used the hashing algorithm. Both simulations provided access to exactly the same function list; however, the hash phone also had some synonyms for function names. Providing too many synonyms would bias the experiment toward the hash handset, so each function name was allowed a maximum of one synonym to compensate for noun–verb and verb–noun transpositioning; for example 'Ringing volume' was also replicated as 'Volume of ring'. These simulations can be found online at [11].

4.3. Procedure

Each subject was given a brief (approximately five minute) explanation of how each handset worked and a demonstration of the type of task they would be expected to perform during the actual experiment. The main purpose of the explanation and demonstration was to ensure familiarity with the computer simulation which required the mouse to press the on-screen buttons. Instruction sheets were also left for the subjects to refer to, should they need reminding of how either simulation worked.

Each subject was then given a set of 24 tasks to complete, 12 with each handset. A typical task is as follows:

Your phone's capacity to store numbers is almost at its limit. Check to see how much space you have left.

The subject would then search for the function which they felt would be used to complete the task. A subject's interaction with the simulation was observed by means of a video splitting cable, which allowed the experimenter to unobtrusively view on their own monitor what was happening on the subject's screen. The output to this second screen was also recorded to aid in the post-experiment interviews.

Subjects were told that they would be given a maximum of two minutes to complete the task or they could choose to give up before the two minutes had elapsed. The idea of self-retirement from a task came from some pre-experiments, where it was clear that users could become locked in a loop within the menu structure and would *never* find the function they sought.

The order in which the tasks were presented to subjects was randomly re-allocated after each subject had completed the experiment. Any given task could have occurred with either the hash-based handset or the menu-based handset. By changing the tasks to be performed on each handset, we ensured that the difficulty experienced in using a given handset was due to the handset itself, not the difficulty of the tasks allocated to that handset.

Furthermore, by having two groups of subjects, it was possible to remove ordering effects by having one group complete their first 12 tasks with the standard simulation and the other group start with the new design simulation. Subjects then swapped and completed the last 12 tasks using the alternative simulation.

The wording of the tasks was carefully chosen so as not to prime subjects and bias them in favour of one particular simulation. For example, the sample task given above does not use the word 'Memory' anywhere as this may favour the hashing simulation which relies on the user entering (Nokia's) key words. By removing these key words from the task description, subjects were required to guess what words they should search for.

After completing the experiment, subjects were given a brief interview which was intended to extract the subjective opinion about using each handset. Subjects were also ask to supply the words they searched for when using the hash handset. (Rather than attempting a disruptive technique such as think aloud, or interrupting subjects after each task, the video recording of the interaction was used to remind subjects in the post-experiment interviews).

4.4. Results

The experimental scenarios completed on the new design took, on average, 9.54 key presses to complete, in comparison to the standard design where 16.52 key presses were required. This is a strongly significant result (repeated measures one tailed t test, $t=3.4$, $df=29$, $p<0.001$) with users requiring approximately 7 *fewer* key presses, on average, to access the functions (Fig. 4).

We discovered a significant difference in mean times between phone types (repeated measures one tailed t test, $t=1.95$, $df=29$, $p<0.03$) (Fig. 5). The overall mean time for the hash phone is 33.42 seconds as compared to 42.02 seconds for the normal handset. This means that regular phone use is taking a quarter

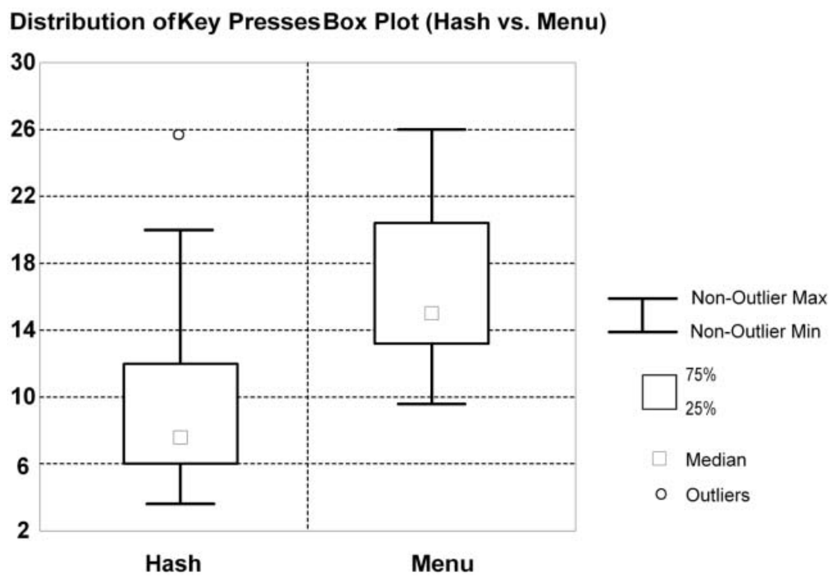


Fig. 4. Box plot showing distributions of key presses used in accessing functions for the hash-based handset and the menu-based

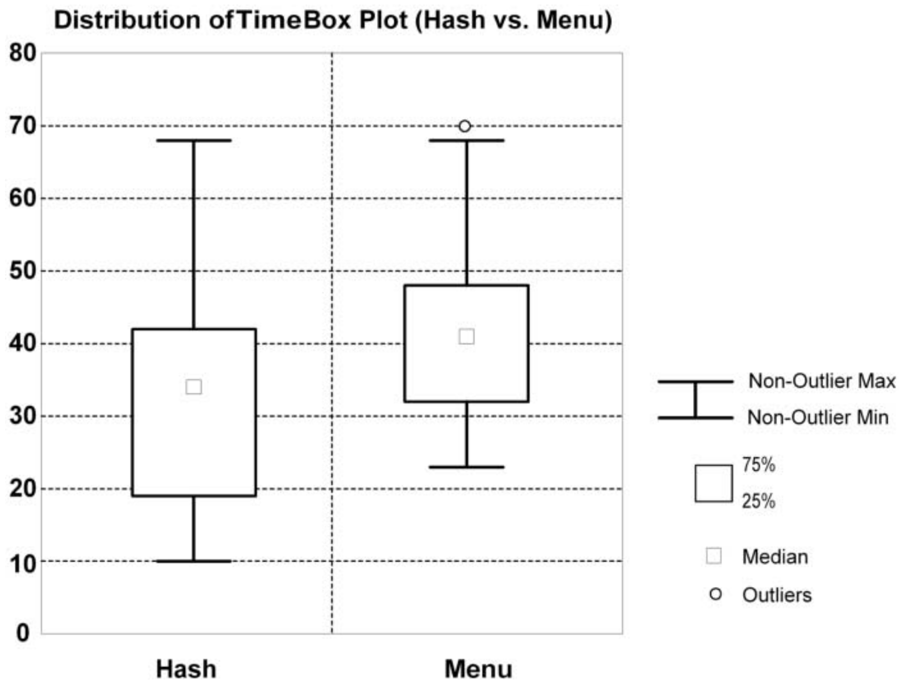


Fig. 5. Box plot showing distributions of the time taken to access functions for the hash-based handset and the menu-based handset.

of the time that hash phones use. This is a considerable improvement for hash phone users.

4.5. Observations

From the subjects' feedback, and from analysing the video recordings we also made the following observations.

- Almost every subject preferred the modified design. This may be due to the fact that it was novel, but many subjects gave convincing reasons such as 'I was able to see all the items in the list— nothing was hidden'.
- The frustration of subjects when using the standard design was obvious. They would

repeatedly become lost as they could not memorize, or visualise, the menu structure. Many subjects became caught in cycles within the menu structure (connections between menu siblings are in a continuous loop) from which they could not escape.

- The ability of the hash handset to scroll through all the functions as a linear list proved very popular with users. In fact, our key press and time scores suffered because three users on the trial ignored the hash-based access to the functions and simply scrolled each time to the function they wanted. These users felt that this was a 'safer', or guaranteed way to find they function they needed. So whilst scrolling may be a slower way to interact, these users willingly sacrificed speed in order to improve their likelihood of finding the function they sought. This 'fail-safe' method of accessing functions is not possible on most current handsets.
- First-time users were confused by having to enter the menu system on the regular handsets. Quite often they would start to scroll without pressing the 'Menu' soft key and become frustrated when they did not see the menu options appearing. Because the hash handset is modeless, these problems were avoided.

5. Design Implications

The results from the experiment support our original decision to base the interface on an optimal data structure. This is not a new idea, having been expressed by Thimbleby as early as 1990 [12], but it is, to our knowledge at least, the first interface that has been created in this way.

Besides giving a quantitative insight into issues such as reductions in key presses, there are side effects from using an algorithmic approach.

- The idea of providing access to data both sequentially and randomly is well understood in computer science, and it seems the same is true for users. In effect, this parallels the notion of providing different interfaces for expert and novice users, something which is not easy to achieve (especially when users migrate from being novice to expert). Our solution provides both hash-based and linear access to the function set in a mutually supporting way!
- The interface behaves in a very consistent way – after all, the interface is generated from a tightly defined source (a computer algorithm).
- It is possible to create shorter user manuals that concentrate on the functionality of the device, not how to access functions. Access to functions is reduced to describing the algorithm used to create the interface, not the menu structure resulting from a designer's thought process.

It is our belief that other interfaces design could benefit greatly from our result. Books such as [7] which list many different data structures and data access algorithms can be used as sources of ideas for interface design. These ideas also provide the designer with analyses and quantitative information about design trade-offs in terms of cost of interaction to access desired functions – this is useful but unusual in user interface design!

6. Future Work

The cellular telephone handsets we examined permitted only key-based interaction. We are now interested in discovering if our results hold true for other embedded computer systems with different physical characteristics. New handsets have new interaction devices, such as scrolling wheels, which are faster than scrolling by key pressing. We are currently building a design tool that will allow interface designers to specify the physical constraints of the device they wish to model and trade off different design solutions to reduce the cost of interaction to its theoretical minimum.

7. Conclusions

A novel interface for cellular telephone handsets based on data structure research was presented. Our analysis showed that our interface should greatly reduce the number of keystrokes required to access a given function. User experiments confirmed this result and also gave us valuable qualitative information: users expressed strongly in favour of the new user interfaces.

Acknowledgements

This work was originally supported by EPSRC grant GR/M14548. Handsets were donated by Orange plc. Thanks to David Nunez in preparing an analysis of the statistics.

References

1. Weaver B. Enhancing ergonomic design for greater appeal at point of sale. In: Proceedings of User Interface Design for Mobile Terminals, 1998, Section 2
2. Youngs E. Evaluating the impact of application, ergonomic and process design on handset success. In: Proceedings of User Interface Design for Mobile Terminals, 1998, Section 1
3. Alexander C. A city is not a tree. Design, 1965; 206: 46–55
4. Nokia. http://www.nokia.com/phones/5110/index.html. Last visited 4 May 2000
5. Miller DP. The depth/breadth tradeoff in hierarchical computer menus. In: Proceedings of the Human Factors Society 25th Annual Meeting, 1981; 296–300
6. Thimbleby H. Analysis and simulation of user interfaces. In: McDonald S, Waern Y, Cockton G (eds.) BCS Conference on Human-Computer Interaction XIV, 2000; 221–237
7. Cormen TH, Leiserson CE, Rivest RL. Introduction to algorithms. MIT Press, 1990
8. Rau H, Skiena S. Dialing for documents: an experiment in information theory. Journal of Visual Languages and Computing 1996; 7: 79–95
9. Tegic. http://www.tegic.com/. Last visited 6 June 2000
10. Gould JD, Boies SJ. Speech filing – an office system for principals. In: Baecker R, Buxton W. (eds.) Readings in Human Computer Interaction. 1987; 8–24
11. Marsden G. Java handset implementation. At http://www.cs.uct.ac.za/~gaz/, last visited 31 May 2001
12. Thimbleby H. User interface design. Addison Wesley, 1990

Correspondence to: Gary Marsden, Department of Computing Science, University of Cape Town, Rondebosch 7701, South Africa. Email: gaz@cs.uct.ac.za

Appendix A

Here are the menu options used in the study. Those not in bold are branch nodes and do not represent an actual function.

- Phone Book
  - **Search**
  - **Add entry**
  - **Erase**
  - **Edit**
  - **Send entry**
  - Options
    - **Type of view**
    - **Memory status**
  - **Speed dials**
- Messages
  - **Inbox**
  - **Outbox**
  - **Write**
  - Message settings
    - **Message centre number**
    - **Message sent as**
    - **Message validity**
  - Common
  - **Delivery report**
  - **Reply via same centre**
- Info Service
  - **Off**
  - **Topics index**
  - Topics
    - **Select**
    - **Add**
    - **Edit**
    - **Erase**
  - **Language**
  - **On**
- **Voice mailbox number**
- Call Register
  - **Missed calls**
  - **Received calls**
  - **Dialled numbers**
  - **Erase recent call lists**
  - Show call duration
    - **Last call**
    - **All calls**
    - **Received calls**
    - **Dialled**
    - **Clear timer**
  - Show call costs
    - **Last call**
    - **All calls**
    - **Clear counter**
  - Call cost settings
    - **Call cost limit**
    - **Show costs in**
- Settings
  - Call settings
    - **Auto redial**
    - **Speed dialling**
    - **Call waiting option**
    - **Own number sending**
  - Phone Settings
    - **Language**
    - **Cell info display**
    - **Welcome note**
    - **Network selection**
  - Security Settings
    - **PIN code request**
    - **Fixed dialling**
    - **Closed user group**
    - **Security level**
    - Change access codes
      - **Change PIN code**
      - **Change PIN2 code**
      - **Change security code**
  - **Restore factory settings**
- Call Divert
  - **Divert all**
  - **Divert when busy**
  - **Divert when not answered**
  - **Divert when phone off**
  - **Cancel all diverts**
- Games
  - **Memory**
  - **Snake**
  - **Logic**
- Calculator
- Clock
  - **Alarm**
  - Settings
    - **Hide**
    - **Set time**
    - **Time format**
- Tones
  - **Incoming alert**
  - **Ring tone**
  - **Volume**
  - **Message alert**
  - **Keypad tones**
  - **Warning and game tones**