Nichols, S. & Ritter, F.E. (1995) A theoretically motivated tool for automatically generating command aliases. In *Proceedings of CHI '95*. 393-400.

Theoretically Motivated Tool for Automatically Generating Command Aliases

Sarah Nichols & Frank E. Ritter
ESRC Centre for Research in Development, Instruction and Training
Department of Psychology
University of Nottingham
University Park
Nottingham NG7 2RD
E-mail: ritter@psyc.nott.ac.uk

ABSTRACT
A useful approach towards improving interface design is to incorporate known HCI theory in design tools. As a step toward this, we have created a tool incorporating several known psychological results (e.g., alias generation rules and the keystroke model). The tool, simple additions to a spreadsheet developed for psychology, helps create theoretically motivated aliases for command line interfaces, and could be further extended to other interface types. It was used to semi-automatically generate a set of aliases for the interface to a cognitive modelling system. These aliases reduce typing time by approximately 50%. Command frequency data, necessary for computing time savings and useful for arbitrating alias clashes, can be difficult to obtain. We found that expert users can quickly provide useful and reasonably consistent estimates, and that the time savings predictions were robust across their predictions and when compared with a uniform command frequency distribution.

KEYWORDS
HCI design tools; Keystroke-Level Model; design problem solving.

INTRODUCTION

Card, Moran and Newell [1] noted that the action is in the design. While some work has gone into tools for design that incorporate their and others' theories (see, for example, Casner and Larkin [3]), most of interface design is still done by hand, and without direct recourse to known HCI theory.

The Rationale for Theoretically Motivated Design

This situation is unfortunate, for there are now numerous HCI results that could be applied, for example, about good and bad HCI interfaces [1, 11], about problem solving and interface use [7, 8], and about perceptual processes [3]. These results offer the opportunity to improve interface design at an early stage. However, it may be beyond the grasp of a designer with a deadline to apply all these results, particularly if the designer has to find the results and then apply them by hand.

The Need For a Tool to Apply Theory to Design

The appropriate approach, it appears to us, is to incorporate theories into a tool or set of tools to either design interfaces or to quickly inspect designs. For an HCI tool to be beneficial, the utility has to be easy to use, cheap, automatic, honest, and complete for the task at hand. The tool must also be flexible to incorporate additional results. This affords fast implementation of psychological theory in an accessible and testable environment.

One often used and easily approached way to improve interfaces is by introducing *aliases* — abbreviations of command names. The use of aliases reduces the work required to execute a command and thus also lowers the possibility of error in expert users (for errors that are simply related to workload). It is also an area where there are known constraints that can inform design, both from empirical research and from HCI theories [1, 7].

Command line interfaces are a worthy area to serve as an example application of such a tool. They are ubiquitous, and as we shall show, the payoff can be quite large. The commands are often needlessly long, and typing mistakes can require a lengthy command to be re-entered. We also chose this type of interface because it is easy to work with and, in this case, has practical use in our daily work.

The use of aliases is not a new concept, but as far as we are aware, there have been no tools put forward to create aliases automatically. It would be useful to have tools that help in HCI related tasks in general — some screen layout tools are available [3] but they are not used much. If this lack of application is caused by the difficulty in applying the theory to a task at hand easily and quickly, a step toward remedying the lack of application of theory to design is to create a tool to make this process easier.

We have created a tool incorporating several known psychological constraints for creating theoretically motivated aliases for command line interfaces. The tool described here has been applied to the command line interface of Soar, a cognitive modelling system [8]. A set of principles for abbreviation method have been incorporated into an alias generating function, and the predicted execution times of the created aliases have been compared with the existing commands using the Keystroke-Level Model.

THE THEORIES IN THE TOOL

We incorporated two sets of psychological theories into our alias design tool. The first predicts how long it will take to enter the command set when they are weighted by frequency. The other is used to derive a set of aliases. They are brought together in a programmable spreadsheet.

The Keystroke-Level Model

We used the Keystroke-Level Model [1] as a measure of design efficiency as it predicts the time taken for a set of commands or aliases to be executed based on sub-task speeds and frequency of tasks. It is a useful and practical simplification of GOMS (Goals, Operators, Methods and Selection) analysis at the level of individual keystrokes [1], when the user's interaction sequence can be specified in detail, as it can here. The Keystroke-Level Model has been shown to predict user performance with between 5 and 20% error [2, 5].

There are several basic assumptions and restrictions associated with the Keystroke-Level Model that this domain and our approach satisfy:

- The acquisition time and execution time for a task are independent.
- The users are experts.
- No account is taken of errors or error recovery.

The total time taken for one task is considered to consist of the time taken to acquire the task (which is not addressed here) and the execution time. The main components of command execution that are considered here are the physical motions required to input the command (in the case of a command line language this is keystrokes) and the number of mental operators that are necessary to remember and correctly execute the commands. Our current model is adaptable for typing speed, but does not include the effect of different typing speeds for different letters. The inclusion of mental operators is governed by heuristics specified by Card et al. [1], specifically rule 2 in Figure 8.2.

An important question to be considered is the increase in burden on memory and mental operators that the introduction of aliases might impose. The Keystroke-Level Model does make several suggestions about how the aliases should be generated consonant with the generation guidelines below. If a rule set is used to generate the aliases, once the initial (very small) rule set has been learned only a single mental operator is required to enter a command alias and that behaviour can become automated more quickly — rather than having to learn by rote each alias, a rule can be applied [1].

The Keystroke-Level model does not specify where the rules or keystrokes come from, or how they are mapped onto the task. This leaves the interface builder with a classic design problem of creating a set of aliases that are easily learned and quickly entered.

Existing Guidelines for Alias Set Generation

While the Keystroke-Level Model does not tell us how to create commands themselves, there are some results that make strong suggestions for how to create aliases of existing commands. There are two main techniques commonly used for abbreviating command names — *truncation* and *contraction* (although other rules have also been used [7]). Truncation involves deleting the last few letters of the original command name, and contraction involves removing letters from the middle and end of the word. In order to avoid confusion, the abbreviation length can be governed by a *minimum-to-distinguish* system, where a sufficient number of letters are used to form the abbreviation to ensure that the abbreviation is unique. However, this means that users must know how many letters are required for each individual command. Research into techniques of abbreviation has shown that abbreviations formed by truncation are more easy to encode (i.e., produce the abbreviation on presentation of the word) than abbreviations formed by contraction [4]. It has been suggested that this is because it is easier to have consistency between commands abbreviated by truncation (e.g., with the rule *always use the first three letters*) as opposed to contraction. A GOMS analysis [7] has also shown that two letter truncation and minimum-to-distinguish are the most efficient forms of abbreviation.

Watts [11] emphasises the need for consistency of abbreviation format within a system, and notes that it is vital to avoid confusion between aliases. If all commands are to be shortened to single letters, then although time would undoubtedly be saved at a keystroke level the memory burden would be greatly increased as the user would have to remember which of the commands were abbreviated to a single letter and which were exceptions. Ehrenreich and Porcu [4] state that if abbreviations are generated by a simple rule then the memory load is greatly reduced. If rules are learned then once the rules are known by the user then their access becomes automated and there is no increase in mental operator time.

Payne and Green [9] have proposed a theory of learnability based on writing a task action grammar (TAG). In their case, different types of subtasks would have different syntax, which would provide a basis for using different abbreviation mechanisms or rules for different types of subtasks. This should decrease the number of clashes and make the resulting aliases more memorable. Aliases presumably would then indicate their category, which would lengthen them. However, if exceptions do occur, then they have to be dealt with using rules similar to the rules above, such as vowel deletion. Since different rule generating mechanisms can be used across subtasks, the user must remember which category the command is in to retrieve the exception rule. An experiment that they performed compared aliases generated for two interfaces, one with a fairly uniform TAG, and with a less uniform TAG. The aliases for the more uniform interface yielded fewer errors, a lower rate of use of a help facility, and a more efficient solution of the problem than the less uniform interface. But their paper by no means states the method supported by Ehrenreich and Porcu [4] produces bad aliases, merely that the command language itself should be regular as well.

In the development and application of this tool we were not prepared to include the redesign of command sets, which the application of TAG encourages, and which Payne and Green note may have to be done before alias creation. We are not sure how easily the Soar commands would lend themselves to being divided up into different types of tasks, as would be necessary if a TAG approach was taken. TAG grammars and semantics (such as destructive commands) could be included as a future extension, by providing a column to indicate a semantic category of each command, and a function that uses that category when automatically creating the alias set.

It can be difficult to apply these principles consistently and time-consuming to implement them by hand. The Keystroke-Level Model notes that command frequency data needs to be gathered, new aliases need to be devised, and in order to assess how useful the aliases are, the time savings need to be computed. Therefore the next step would appear to be to provide a flexible and extendible facility to create and assess aliases automatically.

Dismal – a Motivated Tool for HCI

Dismal [10] is a spreadsheet that was explicitly developed for manipulating psychology data. Dismal is written in GNU-Emacs Lisp, making extensions and modifications such as computations based on the Keystroke-Level Model easy to incorporate. This approach could be used within any programming language, but we prefer a spreadsheet to display the aliases. This visual presentation and the use of functions to compute and display the expected times makes the process easy to follow and provides updates automatically to the designer. We believe that recent versions of commercial spreadsheets now often include a full programming language providing the necessary functionality, but we would be less able to distribute the complete tool, because Emacs is free and can now be run on both UNIX systems and Macs.

The HCI theories were implemented by using two Lisp functions which were added to the Dismal spreadsheet. The first — *key-val* — took a command and calculated the number of mental operators and keystrokes used in the execution of the command. It then used the estimated typing speed to calculate a time prediction for the execution of the command according to the Keystroke-Level Model. The second function — *make-alias* — used the specific rules noted below to automatically generate commands aliases. However, human input was still needed to decide the best way to arbitrate clashes between aliases given the large command set examined here.

EXAMPLE APPLICATION

The command set initially optimised with this tool was taken from Soar [8], a unified theory of cognition realised as a cognitive modelling language. It has previously been command line driven, and over 50 commands are available. In the past year, a Soar Development Environment (SDE) [6] has been developed, which is menu and keystroke driven. The command line interface remains an important part of SDE however, as some members of the Soar community still prefer a command line interface. Also, some Soar users cannot use SDE due to the limitations of the hardware that they use to run Soar.

The ideas behind the alias creation could in fact be applied to any command set, but Soar was a suitable candidate for modification for several reasons.

- The 'alias' command recently added to Soar allows aliases to be added quickly, easily and cheaply.
- The Soar language is currently being used within our local environment and therefore it is worthwhile enhancing the system for both local users and the Soar community world-wide.
- There is potential to get command use frequency information and usability feedback from the local users.
- It is fair to assume that most Soar users can be considered "expert users" and are highly familiar with the commands.
- Some of the commands that exist within Soar are quite long — this suggests that users of the Soar Command Line interface would benefit greatly from the introduction of aliases.

Devising the Aliases

The aliases were devised with the generating function noted above incorporating the alias generation guidelines in the literature (primarily truncation, which means that the abbreviation is the first two letters of the original command). The function aimed to minimise keystrokes while avoiding ambiguity. The characteristics of the command language itself were also considered — particularly with reference to the fact that many of the commands consisted of several words. Therefore, we added the following additional guidelines to create a consistent alias set:

- Include in the alias the first letter of each word in the case of multi-word commands.
- If the length of the command is 5 letters or less, a one letter alias may be provided if clashes do not occur as a result of the new abbreviation. In general this means that already short, abbreviated commands (e.g., *pwd*), do not get shortened to one letter, but short one word commands (e.g., *watch*) do get abbreviated to a single letter.

The value for the mental operator M was taken from Card, Moran, and Newell [1] as 1.35 s. The speed of typing was assumed to be 40 wpm (average non-secretary typist), or 280 ms/keystroke.

Other Aliases Included in Our Set

The alias set generated automatically was augmented for distribution with aliases generated with several additional principles in mind. Computation of time saved, however, was done solely with the main set.

Aliases for experts and novices. While most of our expected users will be experts, every expert used to be a novice, so aliases should be available for use by both. A rule based system for developing aliases is appropriate for both of these categories of users. However, for the purposes of analysis, the times are only applicable to the experts' behaviour, as the Keystroke-Level Model can only be used to predict the speed of an expert completing a task. The alias set is still useful for novices, but this cannot be quantitatively shown using the Keystroke-Level Model.

For experts, once the rules have been learned, no additional knowledge is required in order to be able to use the aliases and so time is instantly saved in the form of keystroke reduction without an increase in mental load from having different abbreviations to remember for different commands

For novices, the meanings of the original commands can still be retained — the aliases are both syntactically and semantically compatible. This can be contrasted with control keystroke command type of aliases, which although reducing time, remove any semantic component of the original command. This is particularly important when considering Soar, which is both a theory and a language, but is also relevant when considering other command line languages.

Aliases — reducing errors. We can reduce how the long commands within Soar can lead to errors in two ways. (a) The number of keystrokes required means that errors are more likely to occur than with shorter commands. Therefore by shortening the command names we expect both to save time and reduce the likelihood of simple typing errors. (b) We included several common misspellings of commands (e.g., *init-saor* for *init-soar*).

Estimations of Task Frequency

The Keystroke-Level Model uses task frequency to balance the time it takes to do each task in a set of unit-tasks. An initial analysis of approximately two hours of actual subject data was performed to compute these frequencies. In this session, commands were used while a specific problem was solved. Perusal of additional transcripts suggested that command usage was highly dependent on the task, there were large individual differences, and in order to generate meaningful frequency data enormous amounts of keystroke logs would be required (we estimate on the order of hundreds of hours). Steps have been taken to log this data in the latest version of the Soar Development Environment [6].

In an attempt to generate useful frequencies more quickly, four expert Soar users, all with more than three years experience working with Soar, were asked to provide frequency estimates for their own use of the original command set. These were easy to provide, although they do not correlate very well with each other (mean pair-wise correlation of 0.49). In future work we expect to validate this approach as an approximation to complete logs.

RESULTS

The automatic function generated 80% of the final aliases, where the rules could be strictly adhered to, and the exceptions were manually adjusted according to frequency information and the conventions shown in Table 1. The complete alias set is shown in Table 2.

Figure 1 shows the estimated time savings based on the time to perform the original command set and the alias set balanced for each individual's frequency distribution. The time taken to execute the original command set is represented by 100%. This total time, based on the normalised command frequency distributions, varied between 245 s (flat frequency) to 458 s (User 3). When the keystroke values of the commands and aliases were calculated and weighted using a uniform command distribution (i.e., assuming all commands were used equally often), it was found that the aliases provided a 55% saving in time over the original command set.

Table 1 - How Exceptions Were Dealt With. The letters here are included as footnotes in Table 2, indicating how the various aliases that could not have automatic aliases created were adjusted.

(a) These commands were already of length <= 3 letters and if they had been shortened any more then clashes would have occurred

(b) The alias *q* rather than *e* is used to prevent accidental exit from Soar as so many other commands have the initial letter e. If *e* is typed then the *echo* command will be executed — this is not a dangerous situation.

(c) The commands *load* and *log* are both short, but are abbreviated by contraction rather than truncation to avoid confusion between the two — if the abbreviation *lo* is typed then a warning is echoed to the screen advising the user to type *ld* or *lg*. The same principle applies to *warnings* and *watch* with a warning being displayed if *wa* is typed.

(d) The command *memory-stats* is shortened to *mems* rather than *ms* to avoid confusion with the already existing Soar command *ms* (which means "list the rules that match"). However, if the user follows the rules then the result is not dangerous.

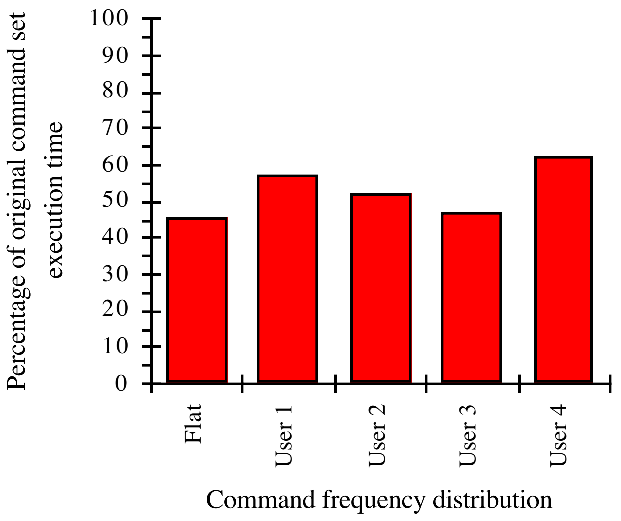

Figure 1 - Comparison of Predicted Benefits of Generated Aliases for Different Command Frequency Distributions.

Although each user had different preferences regarding which commands they used (or thought they used) to perform tasks, when savings were calculated with these frequency estimates, the time savings for individual users for the complete command set ranged between 38 and 53%. When the unchanged command names were omitted from the analysis, the average time savings across all the distributions increased to 62%.

DISCUSSION AND CONCLUSIONS

Overall, this tool appears to be a robust and inexpensive way of applying simple HCI theories to design to reduce command execution time. Aliases can be constructed and tested very easily, and, with the use of an HCI tool, the principles behind the forms of command aliases can be applied routinely and uniformly. Savings estimates can also be documented directly and used to guide design by hand when it is necessary or desirable. The use of a system using guidelines to generate aliases means that the alias forms are easy to learn and can generally be predicted without the particular alias being specified. The local Soar user group has found the alias set to be generally useful, and the aliases have also been distributed to the Soar community at large. Aliases improve the interface at quite modest cost, there appears to be no reason not to take this efficiency gain.

Successes and Implications

Several successes can be drawn from this initial attempt at creating and applying a tool that incorporates an HCI theory. Some of these concern the tool and the theory it incorporates, and others have implications for such theory-based design tools in general.

Comparison with automatic command completion. An analysis using Dismal, Emacs and the existing alias set suggests that automatic command completion would be inferior to this alias set. Command completion would not reduce the keystrokes in this set of commands as much as the aliases do, and would not decrease the mental operators at all.

Implications from the Keystroke Model — The effect of mental operator time. One of the factors that increased the Keystroke-Level Model's time predictions for the original command names was the number of mental operators in the original commands, one for each word. Using single word or acronym aliases are faster not only because they are shorter, but because they require fewer mental operators to perform. In the future, it may be more appropriate to include different types of mental operators for different parts of the task as occurs in the GOMS models [1].

Implications for the Keystroke Model — Task frequency generation and use. Command frequency distributions are used to arbitrate alias clashes and compute expected savings. Ideally, one might assume that these logs should come from actual users. For large command sets, this appears to be unnecessary and considerably expensive. All four of our expert users had different patterns of command use but they all showed a considerable and nearly equal decrease in

Table 2 - Predicted Execution Times for Original Commands and Aliases.
See Table 1 for full explanation of exceptions noted as footnotes. Data imported from Dismal spreadsheet, values rounded to two decimal places.

	Original Command	Predicted time (s)	Alias	Predicted time(s)	Time Saving
1	add-wme	4.87	aw	2.17	56%
2	agent-go	5.13	ag	2.17	58%
3	cd (for chdir)	2.17	cd[a]	2.17	0%
4	chunk-free-problem-spaces	12.42	cfps	2.70	78%
5	d	1.90	d	1.90	0%
6	default-print-depth	9.46	dpd	2.43	74%
7	echo	2.70	e	1.90	30%
8	excise	3.24	ex	2.17	33%
9	excise-all	5.67	ea	2.17	62%
10	excise-chunks	6.49	ec	2.17	67%
11	excise-task	5.94	et	2.17	64%
12	exit	2.70	q [b]	1.90	30%
13	firing-counts	6.49	fc	2.17	67%
14	go	2.17	g	1.90	13%
15	help	2.70	h	1.90	30%
16	init-soar	5.40	is	2.17	60%
17	learn	2.98	le	2.17	27%
18	list-chunks	5.94	lc	2.17	64%
19	list-help-topics	8.64	lht	2.43	72%
20	list-justifications	8.10	lj	2.17	73%
21	list-productions	7.60	lp	2.17	70%
22	load	2.70	ld [c]	2.17	20%
23	log	2.43	lg [c]	2.17	11%
24	matches	3.51	ma	2.17	38%
25	max-elaborations	7.30	me	2.17	70%
26	memory-stats	6.21	mems [d]	2.70	57%
27	ms	2.17	ms [a]	2.17	0%
28	object-trace-format	9.46	otf	2.43	74%
29	p (for print)	1.90	p	1.90	0%
30	pgs	2.43	pgs [a]	2.43	0%
31	preferences	4.59	pr	2.17	53%
32	print-all-help	8.10	pah	2.43	70%
33	print-stats	5.94	ps	2.17	64%
34	ptrace	3.24	pt	2.17	33%
35	pwd	2.43	pwd [a]	2.43	0%
36	quit	2.70	q	1.90	30%
37	r (for run)	1.90	r	1.90	0%
38	remove-wme	5.67	rw	2.17	62%
39	rete-stats	5.67	rs	2.17	62%
40	schedule	3.79	sc	2.17	43%
41	select-agent	6.21	sa	2.17	65%
42	soar-news	5.40	sn	2.17	60%
43	sp	2.17	sp [a]	2.17	0%
44	stack-trace-format	9.19	stf	2.43	74%
45	stats	2.98	s	1.90	36%
46	time	2.70	t	1.90	30%
47	unptrace	3.79	un	2.17	43%
48	user-select	5.94	us	2.17	64%
49	version	3.51	ve	2.17	38%
50	warnings	3.79	wr	2.17	43%
51	watch	2.98	wt	2.17	27%
52	wm	2.17	wm [a]	2.17	0%
TOTAL	(Using flat weighting)	245.17		132.58	55%

execution time when aliases where used. The effect of the new command set appears to be robust across the various frequency predictions, and indeed, even if a flat distribution is used. In order to make this approach more tractable, we suggest using frequency estimates instead.

The Effect on Errors. No users type perfectly, and decreasing command length, if it is consistent, may also decrease errors for expert users (where expert users are those who are familiar with Soar, the command abbreviation rules and the aliases themselves). While we do not have empirical evidence of this yet for our alias set, we can derive a simple prediction. Our alias set reduced the number of keystrokes by 68%. (This is greater than the time savings because the alias generation algorithm removes more keystrokes than mental operators.) The simplest prediction is that the simple typing errors are related to the number of keystrokes, no matter what the users' typing accuracy or speed. Although words are inherently easier to type, these aliases should be fairly easy to type. The semantic component of the command is retained and they are generated in a regular way. This suggests that the number of simple typing errors should significantly decrease as well.

Remaining Problems

While we were generally pleased with the tool and the alias set it generated, three problems remain that need to be acknowledged.

Could the Tool Have Generated More Aliases Automatically? If the rules for devising the aliases had been increased in size then more of the aliases could have been generated automatically. However, the trade-off between increase in the size of the rule set and decrease in the number of exceptions must be considered. If an additional rule had been introduced (e.g., "if truncation to the first two letters of the command leads to a clash, use the first and third letters as the alias instead"), then some of the exceptions would have been automatically generated, but this would have been at the expense of an increase in rules to be learned. It is beyond the scope of this paper to consider the different cognitive burdens that these would impose.

One cannot enumerate all the causes of exceptions — we suspect that this is an open set. We found several here (e.g., usage frequency, clashes, common existing conventions for usage) and do not feel that we have even started to enumerate all the possible causes. This is a problem that is likely to remain, and so while the use of automatic generation should be maximised, the ability for easy manual intervention, such as is provided here with the use of an accessible and flexible spreadsheet, must be retained.

Could the Alias-Generating Tool Have Been More Powerful? As noted above, there are some differences in opinion about which alias generation rules are best, and we have ignored the possibilities of command frequency forcing inconsistencies as well. The point of this paper and tool was not to provide the definitive alias generating function but to show how such a system would work, and provide the means for creating such a function. In other words, the focus should be on our approach, not the specific details of our situation. Since aliases are now cheap and easy to customise, it is not difficult to create additional alias sets for individual users to either adapt our set or create a new set based on their preferred generating rule based on their own cognitive style.

Some points about the Soar aliases do need to be clarified. The Soar command set is a fairly small command set, which has two major implications. First, there were not many clashes formed by the automatic alias generation rules, and second, expert users, for whom aliases are created for, are likely to know all the commands in the set. When a larger command set is being considered, then the need for use of frequency data as a method of arbitration between alias clashes is increased.

Overall effect of improving the command line interface. While we can present a strong case for decreasing typing time, and these aliases appear to increase user satisfaction and reduce errors (although, we have only obtained anecdotal evidence only so far), this result may have a smaller effect on overall task time than might be implied for two reasons. (a) Since the Keystroke-Level model was being used as the analytic tool, the higher level processing was not considered. In the user logs we examined, only a small proportion of time was spent typing (again, this varies largely across users and time, and requires extensive user logs to measure). (b) It is also important to note that several of the commands take arguments, such as files to load or productions to examine. These are not considered here — from a practical point of view, computing their average length is a difficult problem where individual and site differences will reign. Including some estimate remains a necessary step, and the burden that we anticipate it to be suggests that providing them automatically would be a particularly fruitful action to include in the interface.

Future Directions

It is possible that this technique could be directly (although perhaps not simply) extended to analyse menus and other forms of input, as both the Keystroke-Level Model and GOMS analysis could handle this level of complexity. Indeed, the integrated and examinable nature of the menus in the spreadsheet we used (or the other Soar interface) suggests that this would be an appropriate and reachable next step.

Simple Lisp functions were used to implement the command set generation and their analysis. While not a problem in this tool, incorporating additional knowledge and extending the range of tasks that can be examined will become a serious problem quite soon given this representation. A higher level language for representing the constraints on good design and a more complete time prediction system (as could be provided in the future by an extension of Soar itself), will also be necessary steps.

ACKNOWLEDGEMENTS

Support for this work has been provided by the DRA, contract number 2024/004, and by the ESRC Centre for Research in Development, Instruction and Training. Erik Altmann graciously provided access to transcriptions of Soar users. Four anonymous expert Soar users kindly provided us with command frequency estimates. Paul Tingle, Matt Southall and Rob Jones performed a preliminary analysis suggested by Kate Cook that inspired this work. The anonymous CHI reviewers contributed comments that lead to greater clarity.

HOW TO GET DISMAL INCLUDING THESE EXTENSIONS

The latest versions of Dismal and the Soar alias command set are available via anonymous FTP from host unicorn.ccc.nott.ac.uk (128.243.40.7) in the directory "/pub/lpzfr".

REFERENCES

1. Card, S., Moran, T. and Newell, A. The psychology of human-computer interaction. 1983, Hillsdale, NJ: LEA.

2. Card, S.K., Moran, T.P. and Newell, A. The keystroke-level model for user performance time with interactive systems. Communications of the ACM, 1980. 23(7): p. 396-410.

3. Casner, S. and Larkin, J.H. Cognitive efficiency considerations for good graphic design. In Proceedings of the Annual Conference of the Cognitive Science Society. 1989. Hillsdale, NJ: LEA.

4. Ehrenreich, S.L. and Porcu, T. Abbreviations for Automated Syatems: Teaching Operators The Rules, In Directions in Human/Computer Interaction, A. Badre and B. Shneiderman, Editors. 1982, Ablex: Norwood, NJ.

5. Haunold, P. and Kuhn, W. A. Keystroke Level Analysis of a Graphics Application: Manual Map Digitizing. In CHI '94, Human Factors in Computing Systems. 1994. Boston, MA: ACM.

6. Hucka, M, The Soar Development Environment, 1994. Artificial Intelligence Laboratory, University of Michigan: Ann Arbor.

7. John, B.E. and Newell, A. Predicting the time to recall computer command abbreviations. In CHI'87 Conference on Human Factors and Computing Systems. 1987. New York: ACM Press.

8. Newell, A. Unified Theories of Cognition. 1990, Cambridge, MA: Harvard University Press.

9. Payne, S.J. and Green, T.R.G. Task-action grammars: A model of the mental representation of task languages. Human-Computer Interaction, 1986. 2: p. 93-133.

10. Ritter, F.E, Lochun, S, Bibby, P.A. and Marshall, S. Dismal: A free spreadsheet for sequential data analysis and HCI experimentation. In Computers in Psychology '94. 1994. York (UK): CTI Centre for Psychology, U. of York.

11. Watts, R.A. Introducing Interactive Computing. 1984, Manchester: NCC Publications.