AMME - An automatic mental model evaluation to analyze user behavior traced in a finite, discrete state-space

ERGONOMICS, 1993, VOL. 36, NO. 11, 1369–1380

AMME: an Automatic Mental Model Evaluation to analyse user behaviour traced in a finite, discrete state space

MATTHIAS RAUTERBERG

Work and Organizational Psychology Unit,
Swiss Federal Institute of Technology (ETH), ETH Centre,
Nelkenstr. 11, CH-8092 Zurich, Switzerland

Keywords: Usability; Evaluation method; Petri net; State transition space; Complexity; Mental model; User model; Routine task.

To support the human factors engineer in designing a good user interface, a method has been developed to analyse the empirical data of the interactive user behaviour traced in a finite discrete state space. The sequences of actions produced by the user contain valuable information about the mental model of this user, the individual problem solution strategies for a given task and the hierarchical structure of the task–subtasks relationships. The presented method, AMME, can analyse the action sequences and automatically generate (1) a net description of the task dependent model of the user, (2) a complete state transition matrix, and (3) various quantitative measures of the user's task solving process. The behavioural complexity of task-solving processes carried out by novices has been found to be significantly larger than the complexity of task-solving processes carried out by experts.

1. Introduction

The method AMME (Automatic Mental Model Evaluation) was developed to support the cognitive scientist in studying user behaviour. 'Cognitive ergonomics is oriented towards optimizing human–machine systems, according to three types of criteria: characteristics of human cognitive processes, software science knowledge, and knowledge in diverse work domain technologies' (Green and Hoc 1991: 301). Several methods are used to study 'cognitions': questionnaires (Scott *et al.* 1979), ratings of observable behaviour (e.g., 'scoring rationales', McDaniel and Lawrence 1990), protocol analysis (Ericsson and Simon 1984), formal models (Kieras and Polson 1985).

There are different formalisms for constructing user models: MAD (Scapin and Pierret-Golbreich 1990), TAG (Payne and Green 1986), CLG (Moran 1981), CCT (Kieras and Polson 1985), GOMS (Card *et al.* 1983), and various kinds of grammars BNF (Reisner 1981), etc. Using any of these formalisms, the investigator must always categorize user behaviour as appropriate regarding the chosen model and design the more of less pure (= 'error free') user model in a top-down approach based on empirical data. Then she or he can try to test her or his model with empirical data, which has to be cleaned up before testing to become more or less 'error free' (e.g., Churchill 1992). This is often difficult and depends on the expressive power of the used formalism.

If there is a possibility to construct user models in an automatic, bottom-up approach thus avoiding classification of user behaviour as 'defective', then the handling with formal models becomes easy, productive and valid in an empirical sense. Thus, an alternative method to analyse observed empirical process descriptions

without normative constrains like 'defectiveness', etc., are of interest. AMME is a method to construct user models in a clear objective way using full information of the complete process description. It is an automatic, bottom-up approach to construct a formal description of user behaviour. The selected formalism is the net theory. Bauman and Turano (1986) showed, that Petri nets (Petri 1980) are equivalent to formalism based on production rules (like CCT of Kieras and Polson 1985). Finite state transition nets are subsets of Petri nets (Peterson 1981). By modelling the user's knowledge with finite state transition nets Sanderson *et al.* (1989) showed that the state space approach can also be applied in the domain of process control.

Nets are appropriate to describe models in both quantitative and qualitative form. The description of models with quantitative measures allows calculation of applied statistics based on samples of empirical data. Quantitative measures of finite state transition nets are: size (e.g., number of states, number of transitions), complexity (e.g., amount of inter-connectivity), etc.; Qualitative descriptions of nets are: content, form, gestalt, structure (e.g., special pattern), etc. First, the basic idea of the method AMME is presented; the tool and its usage is then described. The last part of this paper presents results of an experimental investigation (Rauterberg 1992a), in which AMME has been applied.

2. Method

What is the main concern of a user interacting with a technical system? The user must build up a mental representation of the system's structure and gain knowledge about the functions of this system with respect to a set of tasks. Furthermore, he must learn the 'language', i.e., a set of symbols, their syntax, and operations connected to them, to evoke interaction sequences (the interactive 'processes') related to task and subtask functions. Thus, the user's representations of the system structure are models of a virtual machine. A 'virtual machine' is defined as a representation of the functionality of a system (functional units and their behaviour). The most important point for the user is the relation between task and machine, rather than the internal structure of the machine's system. Consequently, the task for the human factors engineer is to model a suitable interface as a representation of the virtual machine which can serve as a possible mental representation for the user.

The symbolic representation of the machine system consists of the following elements: (1) objects (things to operate on); (2) operations (symbols and their syntax); and (3) states (the 'system states'). The mental model of the user can be structured in representing objects, operations, states, system structure, and task structure.

2.1. *The basic idea*

Given a finite action space, each state corresponds to a system context, and each transition corresponds to a system operation. A trace (= sequence) of states and transitions in this action space can describe a complete or partial task solving process. Each finite trace in the action space is called a 'process' (see for example figure 1). A task-solving process contains three different kinds of information: (1) all necessary states and transitions itself; (2) the amount of repetition of each necessary state and transition; and (3) the sequential order of all these states and transitions.

Finite state transition nets can be completely described with Petri nets, which have a clear semantic (Peterson 1981). A Petri net is a mathematical structure consisting of two non-empty disjoint sets of nodes, called S-elements and T-elements, respectively,

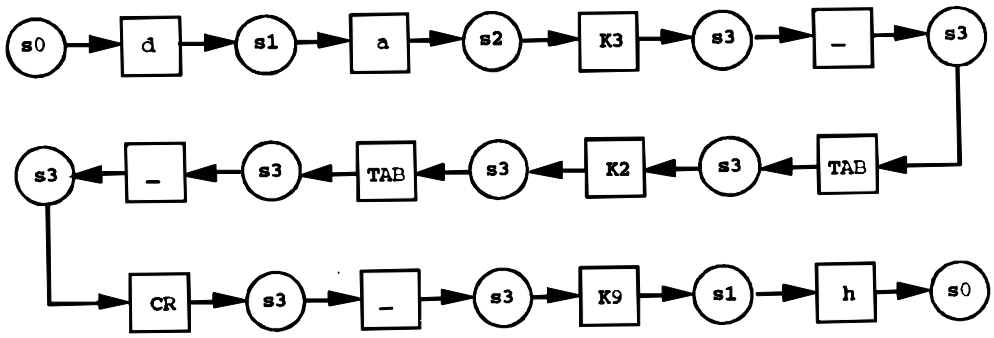

Figure 1. A sequence generated by a user operating a relational database program (see Rauterberg 1992a). The whole process of the shown example is based on 12 transitions (#*AT*) and 12 + 1 = 13 states (#*AS*). The representation of this process is given in a complete ...–>(state)–>[transition]–>(state)–>... description. *Legend*: (*s*0) = main menu state, (*s*1) = module 'data' state, (*s*2) = routine 'browse' state, (*s*3) = wrong input state; [_] = ascii key 'BLANK', [a] = ascii key 'a', [d] = ascii key 'd', [h] = ascii key 'h', [CR] = carriage return, [K2] = function key 'F2', [K3] = function key 'F3', [K9] = function key 'F9', [TAB] = tabulator key.

and a binary relation *F*, called the flow relation. *F* connects only nodes of different types and leaves no node isolated. Petri nets can be interpreted by using a suitable pair of concepts for the sets *S* (signified by a circle '( )') and *T* (signified by a square '[ ]') and a suitable interpretation for the flow relation *F* (signified by an arrow '–>'). The means/activity interpretation allows one to describe the static structure of a system with several active and passive functional components: means (*S*) = real or informational entity, and activity [*T*] = (repeatable) action of a system. The flow relation *F* signifies: [*a*]–>(*m*), the activity *a* (e.g., a system operation) produces means *m* (e.g., a system state); (*m*)–>[*a*], activity *a* uses means *m*.

The main operations (relations) between two nets are abstraction, embedding and folding (Genrich *et al.* 1980). The 'folding' operation is the basic idea of the alternative approach presented in this paper. Folding a process means to map *S*-elements on to *S*-elements and *T*-elements on to *T*-elements while keeping the *F*-structure. The aim of the 'folding' operation is to reduce the elements of an observed empirical task solving process to the minimum number of states and transitions, with the reduced number of elements being the 'performance net'. Folding a task solving process extracts the embedded net structure and neglects the information of the amount of repetition and of the sequential order.

The shortest meaningful part of each process is an 'elementary process': (*s'*)–>[*t'*]–>(*s''*). If the observable behaviour can be recorded in a complete ...–>(state)–>[transition]–>(state)–>... process description (see figure 1), then the analysis and construction of the net structure of this process can be done by counting the number of all different states (#*DS*) and transitions (#*DT*) used, or to mark the frequencies of each state and transition used in the process on a list of all possible states (transitions, resp.). But, if the observable behaviour can only be recorded in a more or less incomplete process description (e.g., ...–>(state)–>[transition]–>[transition]–>...; or ...–>(state)–>(state)–>[transition]–>...), then the analysis and construction of the net structure of this process is difficult: one has to find out the correct state (transition, resp.) between both transitions (states, resp.). Unfortunately, this is the most frequent case in practice. For these cases automatic tool support is

necessary. The program AMME is a tool which offers computer aided support in analysing incomplete recorded processes. This obstacle can be overcome with a description of the complete state transition space in advance, which can be used by the tool AMME to control the correct interpretation of the incomplete process description.

2.2. *Net extraction with AMME*

First of all the investigator has to define a *state list*: a complete description of all states of the interactive system. All possible transitions between different or equal states restrain the action space of the user. One special transition must be taken into account: the 'empty' user action. This is the case when the system automatically changes from one state to another without any user input.

Second, the investigator has to determine a complete action list as a *pre-post state matrix*: a description of all allowed user actions (resp. transitions) changing the actual system state (pre state) to the post state; pre and post states can, but need not be different. The pre-post state matrix is a compact and complete description of all possible elementary processes in the action space. We also designate the pre-post state matrix the 'adjacency matrix' (see figure 3).

The program AMME looks for all elementary processes in the sequence, counts the frequency of each observed elementary process to analyse first order Markov chains, and writes this information in the output file with the 'frequency matrix'. The composition algorithm (the folding operation) of AMME is now able to build up the embedded Petri net combining all observed elementary processes. The result of the folding operation of our example sequence above is the Petri net given in figure 2. This special net with four different states (#*DS*) and nine different transitions (#*DT*) is the minimal net structure we need to reproduce the process given in figure 1. Each 'folded' Petri net is a formal description ('model') of the observed performance of the user's behaviour.

One valuable feature of a net presentation is the possibility for qualitative analysis: one can directly identify specific types of pattern in the net structure. One special pattern, the so called 'corona' pattern around the state *s*3, is given in the net of figure 2. One possible explanation of this pattern could be given as follows: if the cognitive knowledge base of a user is not elaborated enough, and if the interface does not support the user in an appropriate way (e.g., adequate visual feedback), then this user is headed for an interactive deadlock situation. This type of situation is characterized by the fact, that the user does not know, how to leave the actual dialogue context in an appropriate way (in our example state *s*3). In such deadlock situations only trial and error behaviour helps (sometimes!). This trial and error behaviour results in the 'corona' pattern (see figure 2). The behavioural complexity of the 'performance net' with interactive deadlocks increases with the quota of trial and error strategies solving the same task.

2.3. *Input and output files for AMME*

In the context of our research domain (HCI) the tool environment of AMME consists of four different programs: (1) the dialogue system with the logfile recording feature; (2) the analysing program AMME (Rauterberg *et al.* 1992c), which extracts the net of the task and user specific processes and calculates different quantitative aspects of the generated net structure; (3) the Petri net simulator PACE (Dähler 1989); and (4) the analysing program KNOT (Interlink 1991, Schvaneveldt 1990).

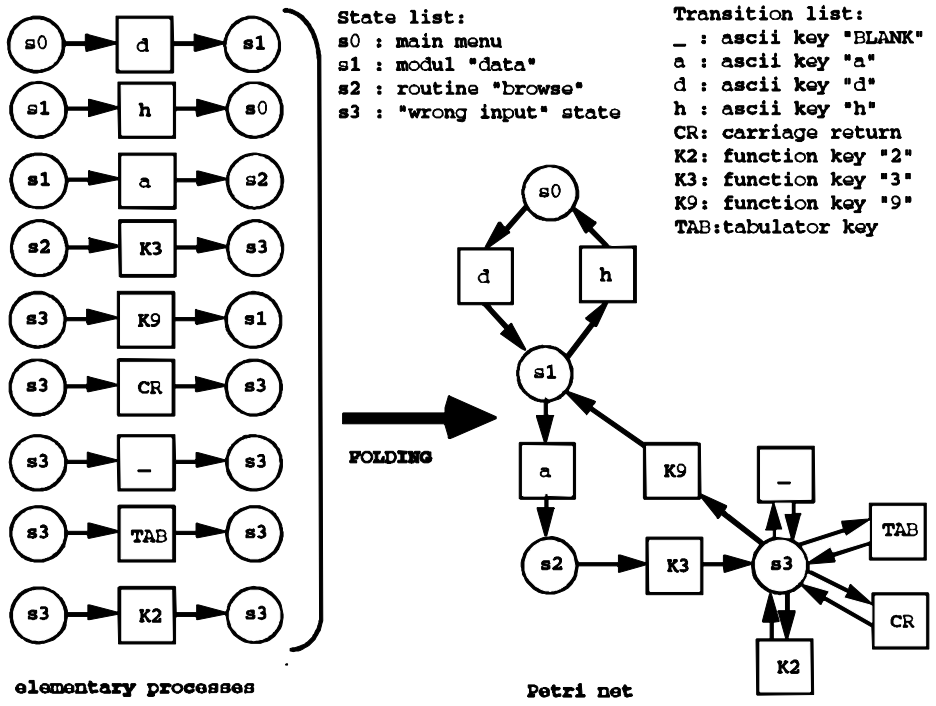

Figure 2. The nine observed elementary processes of the example process shown in figure 1 and the folded Petri net generated by AMME. This folding procedure operates on all elementary processes (*S*0)–>[d]–>(*S*1), ...: all elementary processes of the simple net example (figure 1) are shown on the left side. The semantic labels of all states and transitions are presented on the right side above.

AMME needs two input files: (1) the complete system description on an appropriate level of granularity (the complete state list and the pre-post state matrix); and (2) the process description of a specific individual task solving process corresponding to the granularity level of the system description (see figure 3). The process descriptions can be automatically generated by an interactive system (the 'logfile recording' technique) or hand written by the investigator (e.g., based on protocols of observations).

AMME produces five different output files: (1) a Petri net description file in a special syntactical form for the Petri net simulator PACE; (2) a plain text file with the adjacency matrix for KNOT; (3) a plain text file with the frequency matrix for first order Markov chain analysing software, (4) a plain text file with different quantitative measures of the whole task solving process and the folded net structure; (5) a PostScript file to print the net structure in a graphical form for qualitative pattern matching.

Analysing a set of different process descriptions with AMME is normally an iterative procedure, if the description of the total interactive system is quite large and therefore at the beginning incomplete. With each new process description analysed by AMME the input file with the provisional 'complete' system description must then be updated with 'new' states and/or transitions. After analysing the whole sample of process descriptions in the first trial all process descriptions must be reanalysed in a second trial. This procedure guarantees that the calculation of all quantitative measures of each analysed net is based on the same system description.

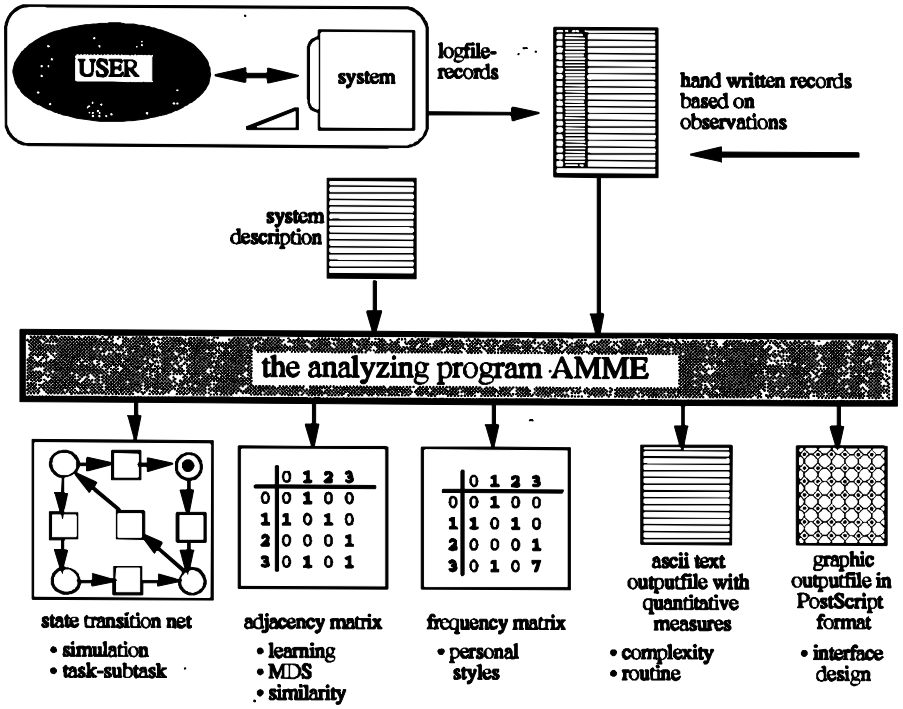

Figure 3. The analysing program AMME requires two input files and generates five different output files: (1) a net description for the Petri net simulator PACE, (2) an adjacency matrix for the Pathfinder software KNOT, (3) a frequency matrix to analyse first order Markov chains, (4) a plain text file with the quantitative measures, and (5) a graphic output of the net structure in PostScript format.

2.4. *Hardware platform for AMME*

The current version of AMME is restricted to descriptions of processes which can be traced in a *finite, discrete state space.* AMME is shareware and available for Apple computers (Mac II with operating system ≥ 7.0) and for IBM or compatible PCs (with MS-Windows ≥ 3.0).

3. Quantifying the task solving process

Our approach is based on the actual observation of users performing a specific task. The key to the interpretation of the process descriptions is a 'map' of the complete task solving domain (action space = state $\cup$ transition space), in which the behaviour of individual processes is traced. The task solving domain in the following example is the whole dialogue net structure of an interactive software program (a relational data base system).

Each observed task-solving process can be quantified with three different measures: task solving time (*#TST*), number of all used transitions (*#AT*), number of all occurring states (*#AS*). Each observed task solving process is always a sequence and can be characterized by $\#AS = \#AT + 1$. These measurements can easily be calculated based on the logfiles.

The extracted net structure can be quantified with the number of different transitions (*#DT*) and the frequency of each transition as well as the number of different states (*#DS*) and the frequency of each state. The accounts of *#DT* and *#DS* depend on the size of the embedded net structure. These measurements are done by AMME.

3.1. *Experimental setting and task description*

In our experiment (Rauterberg 1992a) all users had to play the role of a camping site manager. This manager uses a relational database system with a data base consisting of three data files ('basis relations'): PLACE, GROUP, and ADDRESS.

Novices and experts were classified and selected by their amounts of experiences with electronic data processing in advance. Their experience with electronic data processing was measured with a 115-item questionnaire and with structured interviews. The novice group ($n = 6$) was then instructed in handling of the database system for 1·5 h. The expert group ($n = 6$) already had 1740 h each of experience in handling of the database system. Their total computer experience of approximately 7500 h was the result of their daily work using various types of computers and software systems. The duration of the individual task solving session was about 30 min. Each keystroke with its corresponding time was recorded automatically in a logfile. Each user needed about 1 h for the whole experiment which consisted of 4 tasks in individual sessions. All users had to solve the following four different tasks in operating the database system:

Task 1: 'Please find out how many data records are in the file ADDRESS, in the file PLACE, and in the file GROUP.'

(The user has to activate a specific menu option ('Datafile' in module 'Info' of the menu interface) and to read the file size (solutions: PLACE = 17 data records, GROUP = 27 data records, ADDRESS = 280 data records).)

Task 2: 'Delete only the last data record of the file ADDRESS, the file PLACE, and the file GROUP (sorted by the attribute "namekey").'

(The user has to open (sorted according to the given attribute), select, and delete the last data record (file: PLACE, GROUP, ADDRESS).)

Task 3: 'Search and select the data record with the namekey 'D..8000C O M' in the file ADDRESS, and show the content of all attributes of this data record on the screen. Correct this data record for the following attributes:

State: Germany
Place number: 07
Remarks: Database system dealer can give a demonstration.'

(The user must select a certain data record (file: ADDRESS), update the data record with regard to the three attributes: State, Place number, Remarks.)

Task 4: 'Define a filter for the file PLACE with the following condition: all vacationers arrived on 02/07/87. Apply this filter to the file PLACE, and show the content of all selected data records in the mask browsing mode on the screen.'

(The user must define a filter for the attribute 'arrival data', apply the filter to the data file PLACE, and display the content of each data record found on the screen.)

Each user was instructed to carry out a complete solution for each task as quickly as possible. The investigator checked the completeness of each task solving process.

3.2. *Results and discussion*

A routine task can be identified by a large process description (= long logfile; e.g., $\#AT \gg 10$) and a small size of the embedded net structure (e.g., $\#DT \leq 10$). The user is always running in loops and using the same system operations to solve the task; we designate this kind of task 'routine'. The ratio of 'total number of transitions in the process description' (= the length of the logfile; $\#AT$) to the 'number of different

Table 1. Results of the two factor analysis of variances of the 'degree of routinization' (#*R*) of novices and experts for four different tasks. The variables 'no. of different states (#*DS*)' and 'task-solving time (#*TST*)' are used as covariates.

Dependent variable #*R*: 'routinization degree'	Covariate: without			Covariate #*DS*: 'no. of different states'			Covariate #*TST*: 'task-solving time'		
Source of variation	df	*F*	*p*	df	*F*	*p*	df	*F*	*p*
'Expertness'	1	6·69	0·013	1	9·88	0·003	1	1·40	0·243
'Tasks (1–4)'	3	14·51	0·001	3	15·44	0·001	3	11·16	0·001
'Expertn. '⊗' tasks (1–4)	3	2·22	0·101	3	2·05	0·054	3	1·70	0·182

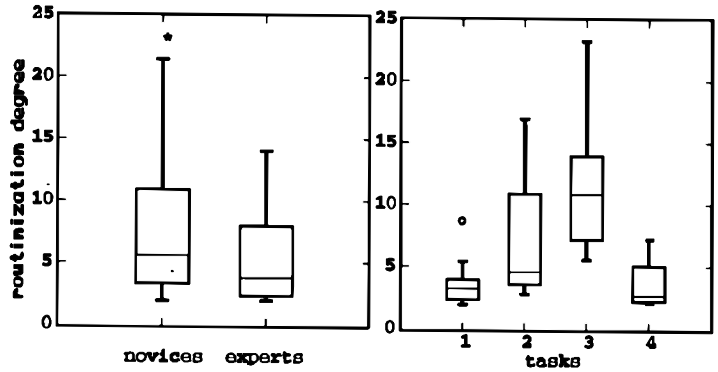

Figure 4. These two Box-and-Whisker Plots show the 'degree of routinization' (#*R*) for 'novices–experts' and 'tasks 1–4' ($n = 48$) (separately). The box in each case covers the middle 50% of the data values, between the lower and upper quartiles. The extension lines indicate the extremes, while the central line marks the median.

transitions in the folded net' (#*DT*) is a good measure of the 'degree of routinization' (#*R*).

$$\text{'degree of routinization'}: \quad \#R = \#AT/\#DT \tag{1}$$

This measure combines the information of the total amount of repetition of each transition (#*AT*) with the information of all necessary transitions (#*DT*). The information of the sequential order of all transitions is neglected.

We calculated the 'degree of routinization' (#*R*) of these four different tasks and analysed this measure with a two factor analysis of variances (table 1). A statistically significant difference between novices and experts and between the four different tasks (figure 4) was found. The difference between novices and experts can be fully explained using the 'task-solving time' as a covariate in the analysis of variances, but not the difference between the four tasks (table 1, last column). This is a strong indication, that the 'degree of routinization' (#*R*) is a task feature rather than a characteristic of users.

The 'editing' Task 3 in our investigation was found to be the task with the largest #*R* (figure 4). The size of the observable task solving process of task 3 is on average 10 times larger than the underlying net structure.

4. Quantifying the behavioural process

One important difference between novices and experts is the complexity of their mental model (Bainbridge 1991: 343). The novices have a simple mental structure, so they have to behave in a more heuristic manner to operate an interactive system. On the other hand, the structure of the mental model of the experts are more or less correct representations of the system structure, so they can behave in a straightforward fashion to solve given tasks. We therefore assume that the mental model of an expert is more comprehensive than that of a novice. This assumption can be tested by comparing the complexity of the observable behaviour of novices and experts.

We call the complexity of the observable behaviour the 'behaviour complexity (*#BC*)'. This behaviour complexity can be estimated by analysing the recorded concrete task solving process ('logfiles'). The complexity of a given work system (the 'action space') we call 'system complexity (*#SC*)'. The account of *#SC* is given by the complete pre–post state matrix, which contains all possible elementary processes. The complexity of the observable behaviour (*#BC*) of novices should be on average larger than the complexity (*#BC*) of the observable behaviour of experts.

4.1. *Definition of complexity*

To measure complexity we introduce the C_{cycle} metrics (the 'cyclomatic number') of McCabe (1976). C_{cycle} is a measure to calculate the number of linear independent cycles of a plane and coherent net. C_{cycle} is a useful quantity that measures the amount of interconnectivity (complexity). The advantages and disadvantages of four different quantitative measures of complexity in the context of our empirical investigation have been discussed on different occasions; C_{cycle} proved to be the most effective (Rauterberg 1992b).

The complexity measured with C_{cycle} is defined by the difference of the total number of connections (*#T*: transition) and the total number of states (*#S*: state). The parameter P is a constant that corrects the result of Formula 1 in the case of a sequence ($\#T - \#S = -1$); the value of P in this context is 1. The semantic of C_{cycle} can be described by the number of 'holes', 'loops', or 'cycles' in a net.

$$\text{'cyclomatic number':} \qquad C_{cycle} = \#T - \#S + P \qquad \text{with } P = 1 \tag{2}$$

The measure C_{cycle} of the net example in figure 3 is $\#DT - \#DS + P = 9 - 4 + 1 = 6$; the complexity of the observed task solving sequence shown in figure 1 is $\#AT - \#AS + P = 13 - 12 + 1 = 0$; for each sequence C_{cycle} is zero (a sequence has no cycles!).

4.2. *Results and discussion*

Cognitive complexity has been defined as 'an aspect of a person's cognitive functioning, which at one end is defined by the use of many constructs with many relationships to one another (complexity), and at the other end by the use of few constructs with limited relationships to one another (simplicity)' (Pervin 1984: 507). Transferring this broad definition to human–computer interaction could imply that the complexity of the user's mental model of the dialogue system is given by the number of known dialogue contexts ('constructs') on one hand, and by the number of known dialogue operations ('relationships') on the other. In order to measure the complexity of a mental model which generates an observable process, a 'mapping' procedure from the observable process description to the embedded structure of this process is necessary. This 'mapping' procedure can be done with the folding operation in the context of Petri

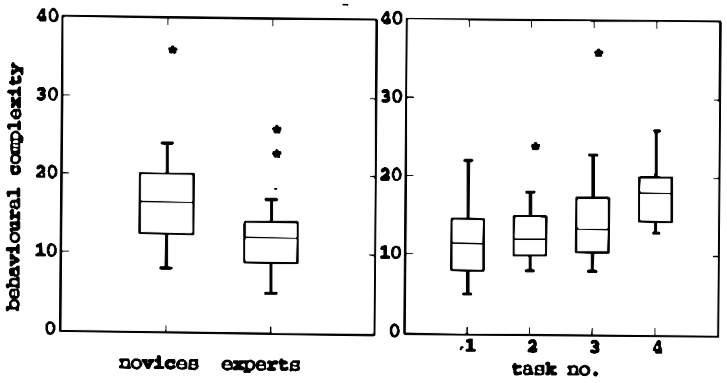

Figure 5. These two Box-and-Whisker Plots show the 'behavioural complexity' (*#BC*) for 'novices–experts' and 'tasks 1–4' ($n = 48$) (separately). The box in each case covers the middle 50% of the data values, between the lower and upper quartiles. The extension lines indicate the extremes, while the central line marks the median.

Table 2. Results of the two factor analysis of variances of the 'behavioural complexity' (*#BC*) of novices and experts for four different tasks. The variables 'no. of different states (*#DS*)' and 'task-solving time (*#TST*)' are used as covariates.

Dependent variable *#BC*: 'behavioural complexity'	Covariate: without			Covariate *#DS*: 'no. of different states'			Covariate *#TST*: 'task-solving time'		
Source of variation	df	*F*	*p*	df	*F*	*p*	df	*F*	*p*
'Expertness'	1	10·34	0·003	1	15·35	0·001	1	0·07	0·792
'Tasks (1–4)'	3	3·25	0·032	3	1·34	0·276	3	6·07	0·002
'Expertn. '⊗' tasks (1–4)	3	0·32	0·810	3	0·49	0·693	3	1·12	0·354

nets. The 'behaviour complexity' of the folded net (based on the 'cyclomatic number') is defined as:

$$\text{'behaviour complexity'}: \quad \#BC = \#DT - \#DS + 1 \tag{3}$$

We can show, that the complexity of the observable task solving behaviour (the 'behaviour complexity'; *#BC*) of novices is significantly larger than the complexity of the observable behaviour of the experts (see figure 5 and table 2). The difference between novices and experts can be fully explained using the 'task-solving time' as a covariate in the analysis of variance, but not the difference between the four tasks (table 2, last column). This is a strong hint, that the 'behaviour complexity' (*#BC*) depends only on task features.

The Spearman rank correlation (r) between the 'total computer experience' (measured with the questionnaire) and the mean of *#BC* (averaged over tasks 1–4) is $r = -0{\cdot}682$ ($n = 12$; $p \leq 0{\cdot}02$). This important result indicates, that the complexity of the observable behaviour correlates negatively with the complexity of the cognitive structure (the 'mental model'). So, the complexity of the necessary task knowledge can be either observed and measured with *#BC* or is embedded in the cognitive structure. If the cognitive structure is too simple, then the concrete task-solving process must be carried out with a lot of heuristics or trial and error strategies. Learning how to solve a specific task with a given system means that *#BC* decreases and the complexity of the mental model increases (*#CC*).

5. Conclusions

In general a user interaction with a system can be described with a finite, discrete state transition net, and behaviour patterns can be traced with a sequence of states and transitions $(s')->[t']->(s'')->[t'']->(s')->[t']->(s'')->[t'']->\ldots$ To investigate the interaction processes with this kind of more or less complex systems, support of special tools is necessary. AMME is a computer-aided tool which analyses incomplete sequences of states and transitions, and generates the underlying net structure and different measures of complexity of the interactive process described by the state transition sequence.

In comparing the size of the observed task solving process with the size of the embedded net structure we are able to measure the 'degree of routinization' ($\#R$) of tasks. This measure $\#R$ allows us to differentiate between different types of tasks in a clear quantitative way.

Based on the empirical result that the complexity of the observable behaviour of novices is larger than the complexity of experts, we conclude that the behavioural complexity is negatively correlated with the complexity of the mental model. Thus it is possible to estimate the cognitive complexity based on the measurement of the behavioural complexity, the measurement of the system complexity and the measurement of the task complexity (for a more detailed discussion see Rauterberg 1992b).

Acknowledgements

The preparation of this paper was supported by the German Secretary of State for Research and Technology (BMFT-AuT programme) grant number 01 HK 706-0 as part of the research project 'User-oriented software development and interface design (BOSS)' and the ADI Software Inc. in Karlsruhe, Germany.

References

BAINBRIDGE, L. 1991, The 'cognitive' in cognitive ergonomics, *Le Travail humain*, **54**, 337–343.

BAUMAN, R. and TURANO, T. A. 1986, Production-based language simulation of Petri nets, *Simulation*, **47**, 191–198.

CARD, S. K., MORAN, T. P. and NEWELL, A. 1983, *The Psychology of Human–Computer Interaction* (Lawrence Erlbaum Associates, Indiana).

CHURCHILL, E. 1992, The formation of mental models: are 'device instructions' the source? in G. van der Veer, M. Tauber, S. Bagnara and M. Antalovits (eds) *Human–Computer Interaction: Tasks and Organisation* (CUD, Roma), 3–16.

DÄHLER, J. 1989, *The Petri net simulator PACE* (Pace Inc., Neptunstrasse 16, CH-8280 Kreuzlingen, Switzerland).

ERICSSON, K. A. and SIMON, H. A. 1984, *Protocol Analysis* (The MIT Press, Cambridge, MA).

GENRICH, H. J., LAUTENBACH, K. and THIAGARAJAN, P. S. 1980, Elements of general net theory, in W. Bauer (ed.) *Lecture Notes in Computer Science vol. 84 'Net Theory and Applications'* (Springer, New York), 21–163.

GREEN, T. R. G. and HOC, J. M. 1991, What is cognitive ergonomics? *Le Travail humain*, **54**, 291–304.

INTERLINK, INC. 1991, *The KNOT software* (P.O. Box 4086 UPB, Las Cruces, NM 88003, USA).

KIERAS, D. E. and POLSON, P. G. 1985, An approach to the formal analysis of user complexity, *International Journal of Man–Machine Studies*, **22**, 365–394.

MCCABE, T. 1976, A complexity measure, *IEEE Transactions on Software Engineering*, **SE-2(6)**, 308–320.

MCDANIEL, E. and LAWRENCE, C. 1990, *Levels of Cognitive Complexity: An Approach to the Measurement of Thinking* (Springer, New York).

MORAN, T. P. 1981, The command language grammar: a representation for the user interface of interactive computer systems, *International Journal of Man–Machine Studies*, **15**, 3–50.

PAYNE, S. J. and GREEN, T. G. R. 1986, Task–action grammars: a model of the mental representation of task languages, *Human–Computer Interaction*, **2**, 93–133.

PERVIN, L. A. 1984, *Personality* (John Wiley & Sons, New York).

PETERSON, S. J. 1981, *Petri Net Theory and the Modeling of Systems* (Prentice Hall, Englewood Cliffs).

PETRI, C. A. 1980, Introduction to general net theory, in W. Bauer (ed.) *Lecture Notes in Computer Science vol. 84 'Net Theory and Applications'* (Springer, New York), 1–19.

RAUTERBERG, M. 1992a, An empirical comparison of menu-selection (CUI) and desktop (GUI) computer programs carried out by beginners and experts, *Behaviour & Information Technology*, **11**, 227–236.

RAUTERBERG, M. 1992b, A method of a quantitative measurement of cognitive complexity, in G. van der Veer, M. Tauber, S. Bagnara and M. Antalovits (eds) *Human–Computer Interaction: Tasks and Organisation* (CUD, Roma), 295–307.

RAUTERBERG, M., HOFMANN, J., LEUCHTER, C. and RUDNIK, J. 1992c, AMME: an automatic mental model evaluator, Technical Report, Work and Organizational Psychology Unit, Swiss Federal Institute of Technology, Zurich.

REISNER, P. 1981, Formal grammar and human factors design of an interactive graphics system, *IEEE Transactions on Software Engineering*, **SE-7(2)**, 229–240.

SANDERSON, P. M., VERHAGE, A. G. and FULD, R. B. 1989, State-space and verbal protocol methods for studying the human operator in process control, *Ergonomics*, **32**, 1343–1372.

SCAPIN, D. L. and PIERRET-GOLBREICH, C. 1990, Towards a method for task description: MAD, in L. Berlinguet and D. Berthelette (eds) *Work with Display Units 89* (Elsevier, Amsterdam), 371–380.

SCHVANEVELDT, R. W. 1990, *Pathfinder Associative Networks—Studies in Knowledge Organization* (Ablex Publishing, Norwood).

SCOTT, W. A., OSGOOD, D. W. and PETERSON, C. 1979, *Cognitive Structure: Theory and Measurement of Individual Differences* (John Wiley & Sons, New York).

WASSERMAN, A. I. 1985, Extending state transition diagrams for the specification of human–computer interaction, *IEEE Transactions on Software Engineering*, **SE-11(8)**, 699–713.