

A Petri Net Based Analysing and Modelling Tool Kit for Logfiles in Human-Computer Interaction

Matthias Rauterberg

Swiss Federal Institute of Technology (ETH)
Nelkenstr. 11, CH-8092 Zurich, Switzerland
Tel: +41-1-632 7082, Email: rauterberg@ifap.bepi.ethz.ch
Internet: <http://www.ifap.bepi.ethz.ch/~rauterberg/>

Abstract

To support the human factors engineer in designing a good decision support system, a tool kit has been developed to analyze the empirical data of the interactive decision behaviour described in a finite discrete state space (e.g., human-computer interaction). The observable sequences of decisions and actions produced by users contain much information about (1) the mental model of this user, (2) the individual problem solution strategies for a given task and (3) the underlying decision structure. The presented analysing tool kit AMME can handle the recorded decision and action sequences and comes up automatically (1) with an extracted net description of the task dependent decision model, (2) with a complete state transition matrix, and (3) with different quantitative measures of the decision behaviour.

Introduction

Several methods are developed and used to study cognition's: questionnaires (Scott, Osgood and Peterson 1979), protocol analysis (Ericsson and Simon 1984), formal models (Kieras and Polson 1985), scoring rationale of observable behaviour ('interpretative exercise' McDaniel and Lawrence 1990). "Cognitive ergonomics is oriented towards optimising 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, p.301). The method presented in this paper was developed to support the 'cognitive scientist' in modelling human decision behaviour.

The normal design cycle to construct a formal model of human behaviour is a task dependent top down approach (see de Haan, van der Veer and van Vliet 1991). Given a top down constructed model, it has to be validated with empirical data (e.g., Schröder et al. 1990). In this paper we present an alternative automatic bottom up approach to construct a formal description of user decision behaviour. The presented method constructs user models in a bottom-up, automatic and clear objective way. The formalism we selected is the net theory. To model the user's knowledge with finite state transition nets Sanderson, Verhage and Fuld (1989) showed, that the state space approach works in the domain of process control, too. The scope of this paper does not enclose approaches based on questionnaires (Scott, Osgood and Peterson 1979), or other scoring rationale techniques (McDaniel and Lawrence 1990). The presented approach is an integrated modelling environment based on Petri nets (see Reising 1992, Jones 1993).

On Models and Modelling in Human-Computer Interaction

Geoffrion (1989) describes modelling processes on four different levels: modelling traditions, modelling paradigms, model classes, and specific models. Our modelling tradition is human-computer interaction and cognitive ergonomics. All classes of mental and/or task models describing user behaviour are our *modelling paradigm*. All interactive processes generated by users over different tasks with a concrete interactive system forms our *model class*. In our context a *specific model* is a concrete user solving a specific task with a real database system (cf. Rauterberg 1995).

Dolk (1993) differentiates between three main schools of model representation: structured modelling, logic modelling, and graph grammars. Our approach can be subsumed under graph grammars. There are different formalisms for constructing models of cognitive processes: Cognitive Complexity Theory (Kieras and Polson 1985), GOMS (Card, Moran and Newell 1983), and different kinds of grammars BNF (Reisner 1981), EBNF (Reisner 1984) etc. Using any of these formalisms the investigator must always design the pure (more or less 'error free') user and/or task model in a top down approach based on a task analysis (Booth 1991). Then he can try to prove his model with 'error free' data (e.g., Churchill 1992). This is often difficult, time consuming, and expensive (Kellogg and Breen 1990). If there is a possibility to construct the mental models of users in an automatic, bottom up approach based on automatic recorded logfiles, then the handling with mental models becomes easy and productive.

Oberquelle (1984) shows, that several notions of 'model' are in use: (1) a model of an axiom system, (2) a known system with structure and behaviour analogous to the system under consideration, (3) a prototypical system in the sense of 'model farm', or (4) an abstract description of the relevant aspects of a system.

Class (1) models are normally given in terms of mathematics. We do not go into more detail of this class type. A class (2) model can be called an 'exact analogy' in the sense of a special case of a metaphor. Class (3) models are developed in the context of tutorial systems (e.g., Schröder et al. 1990). Class (4) models are useful to classify and explain phenomena of the modelled system. We present in this paper a method to generate class (4) models.

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 sym

bols, their syntax, and operations connected to them, to evoke interaction sequences (the interactive 'processes') related to task and sub task functions. So, 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, and not so much 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, decision and task structure.

Modelling with Petri nets

A net can be described as a mathematical structure consisting of two non-empty disjoint sets of nodes (S-elements and T-elements), and a binary flow relation (F) (Peterson 1981). The flow relation links only different node types and leaves no node isolated (Petri 1980). Petri nets can be interpreted in our context 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 '->').

A net only with this three net elements (S, T, F) can be interpreted as a *causal net*, that models only the causal relations. If we want to simulate dynamic processes in time, then we need a new element to signify activities. This additional net element is called a *token*.

Condition-event nets run with unmarked tokens. An event occur if certain preconditions are fulfilled, after the occurrence, certain post conditions hold. A token in the circle of a S-element means that the corresponding condition holds. If we need a distinction between different types of token, then we mark them in a characteristic way.

Place-transition nets run with marked tokens. This three net types are often called Petri nets. Törn (1985) gives us an overview of four important advantages of Petri nets:

- Petri nets are theoretically well founded.
- Petri nets are well suited for describing asynchronous concurrent processes.
- A Petri net simulator tool is simple and easy to learn.
- Both top down and independent modelling (including validation) of different aspects is possible.

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) interaction with or action of a system. The flow relation F means: [a] -> (m), the activity [a] (e.g., a user action) produces means (m) (e.g., a system state); (m) -> [a], activity [a] uses means (m) (Oberquelle, Kupka and Maass 1983).

Bauman and Turano (1986) showed, that Petri nets are equivalent to formalism based on production rules (like CCT

of Kieras and Polson 1985). In this sense, the presented approach can be subsumed under 'logic modelling', too.

Applying the path algebra approach of Alty (1984) to analyze the adjacency matrix, we can get all "elementary paths in the network (e.g., paths which do not traverse an arc more than once). This algebra exhibits closure and the closure matrix gives all possible elementary paths between nodes" (Alty 1984, p. 125).

The Method

The Basic Idea

The main operations (relations) between two Petri nets are abstraction, embedding and folding (Genrich et al. 1980).

The Folding Operation

The *folding operation* is the basic idea of the approach presented in this paper. Folding a process means to map S-elements onto S-elements and T-elements onto T-elements while keeping the F-structure. The result is the structure of the performance net (see Figure 1). Each state corresponds to a system context, and each transition corresponds to a system operation. This sequence is called a 'process'. An *elementary process* is the shortest meaningful part of a sequence: (s') -> [t'] -> (s'') with s' as the prestate and s'' as the poststate of transition t'.

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 logical 'task structure'. Folding a task solving process extracts the embedded net structure and neglects the information of the amount of repetition and of the sequential order of actions in the process.

Complete versus Incomplete Task Solving Descriptions

If the observable behaviour can be recorded in a complete ...-> (state) -> [transition] -> (state) ->... process description, then the analysis and construction of the net structure of this process are simple: You have only to count the number of all different states and transitions used, or to mark on a list the frequencies of each state and transition used in the process.

But, if the observable behaviour can only be recorded in an *incomplete* (e.g., ...-> (state) -> [transition] -> [transition] ->... or ...-> (state) -> (state) -> [transition] ->...) process description, then the analysis and construction of the net structure of this process are difficult. You have to find out the correct state (transitions, resp.) between both transitions (states, resp.). Unfortunately, this is the most frequent case in practice, because most of all existing systems do not have a complete internal representation of all possible system states.

For all interactive systems with no internal state representation we need automatic tool support. For these cases we developed a tool kit, that gives us the possibility to analyze any processes with an incomplete process description, that are generated by finite state transition nets. The price we have to pay is the definition of a complete state transition description of all relevant transitions beforehand.

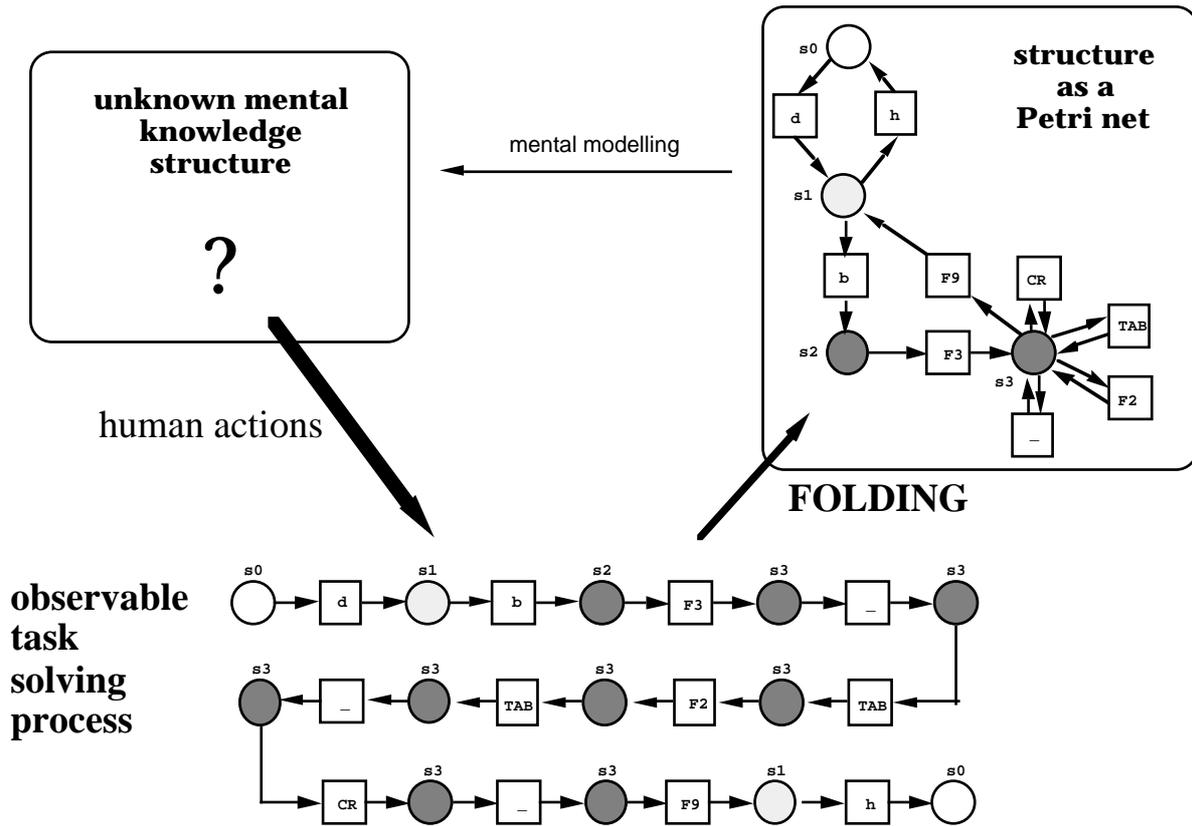


Figure 1: The 'folding' operation as the basic idea of AMME.

The Petri Net Construction Process

The Architecture of the Analysing Tool Kit

The whole system of our analysing tool kit consists of seven different programs (see Figure 2):

- (A) An interactive *dialog system* with a logging feature generates the task solving process description. This process description should be automatically transformed to a logfile with an appropriate syntactical structure. A logfile can also be hand written by the investigator (e.g., based on protocols of observations).
- (B) The *net generation program* AMME (1996) extracts the net of interactive process description and calculates different quantitative measures of the generated net. AMME needs three input files: (1) the interactive process description (the logfile), (2) a complete system description on an appropriate level of granularity (the state list and the pre-poststate matrix), and (3) a support file for the graphic output ("defaultp.ps"). AMME produces five different output files: (1) a protocol file (*.pro) with different quantitative measures of the process at all and of the extracted net; (2) a Petri net description file (*.net) in a readable form for the Petri net simulator PACE (1995); (3) a plain text file (*.ptf) with the connectivity matrix for KNOT; (4) a plain text file (*.mkv) with the probability matrix for the Marcov chain analysing software SEQUENZ, and (5) a PostScript file (*.ps) to print the net graphic for pattern matching 'by hand'.

- (C) The *Petri net simulator* PACE (1995) is a commercial product; PACE is implemented in Smalltalk 80 and consists of a graphic editor and an interactive simulator with graphic animation. PACE can deal with hierarchical nets, refinement of T- and S-elements, timed Petri nets, and stochastic Petri nets. Smalltalk 80 standard classes are available for token attributes.
- (D) With the *net analysing program* KNOT (Interlink 1991) we can compute the similarity between pairs of nets (Schvanefeldt 1990). With the multidimensional scaling (MDS) module of KNOT (Kruskal non-metric MDS algorithm) we can compute a MDS solution for any set of nets.
- (E) The Marcov analysing software SEQUENZ (Schmid and Meseke 1991) presents a method for comparing sequences produced by computer users. In a first step the sequences are transformed into first-order Marcov-chains. Similarity between these lattices can be directly calculated by summation of the differences between lattice-cells. The resulting distances provide input data for MDS models, too.
- (F) Any standard Postscript interpreter (e.g., Ghostscript) can read and print the output file *.ps.
- (G) Any text processing software can read the pure ASCII file *.pro.

The current version of AMME is restricted to process descriptions that can be traced in a *finite, discrete state space* with an upper limit of different states. Another restriction is the constrained syntax of the logfiles as input for AMME.

To transform a given logfile into the necessary form, several tools can be used: Coco/R (Mössenböck 1990), YACC (Johnson 1975), or any other tool that can replace text strings by other strings. AMME is shareware and available for IBM or compatible PCs (with MsWindows ≥ 3.0).

The Data and Program Structure of AMME

We choose an open data structure to maximise the flexibility of AMME. The only restriction is the number of dialog states (up to a fix number of definable state identifiers). Each identifier can be an alpha-numeric name up to 60 characters. All defined transitions are organised in an open list (the limitation for this list depends only on the actual amount of main memory). The program structure of AMME can be described as a double compiler: one syntax checker and parser for the logfile, and another syntax checker and parser for the system description file.

The User's Logfile

As outlined in Figure 2 the observable task solving processes are recorded in a logfile. AMME expects the logfile content in a given syntactical form. This syntax is very

simple and allows only seven different event types per line with different parameters: "LOG_KEYBD", "LOG_MENU", "LOG_MESSAGE", "LOG_ERROR", "LOG_HELP", "LOG_USRTIME", and "LOG_SYSTIME".

These seven different event types are helpful for the investigator to structure the system description file in a readable form; they are not really necessary for this analysing method itself. The both events "LOG_USRTIME", and "LOG_SYSTIME" have no representation in the system description file, but both time stamps are summed up separately during the parsing process.

Analysing a set of different task solving process descriptions with AMME is normally an iterative procedure. With each new process description analysed, sometimes the file with the system description must be updated with unconsidered and therefor new states and/or transitions. After analysing the whole set of process descriptions in the first trial, all process descriptions must be re-analysed in a second trial. This procedure guaranties, that the calculation of all quantitative measures of each analysed net is based on the same system description.

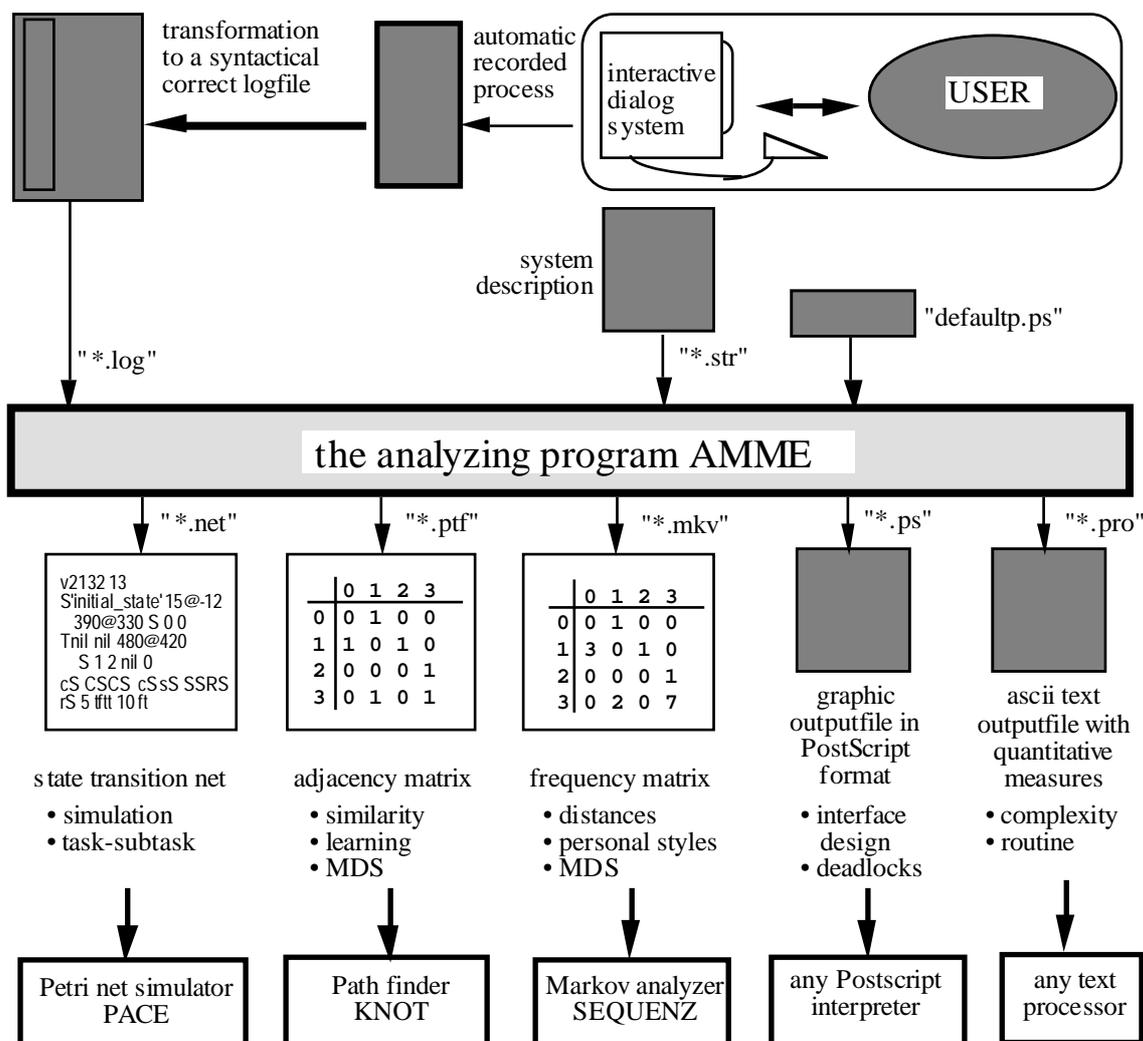


Figure 2: An overview over all involved programs and files of the analysing tool kit.

The System Description File

First, the investigator has to define a state list: a complete description of each relevant state of the investigated system. The definition of a relevant 'system state' is given by the concrete system's behaviour or by the interpretation of the investigator. Each output sequence, each error message or any other distinguishable situation can be a relevant state. The first thing an investigator has to do is to choose the appropriate 'granularity' level. The highest possible level is each keypress or mouse click itself. This level is only of interest if one try to model user behaviour on the keystroke level (see KLM in Card, Moran and Newell 1983).

The next aggregation levels are different combinations of several keystrokes to an operator that trigger the corresponding transition. All keystrokes that are not of primary interest can be read over with the special operator "ALL".

Second, the investigator has to define the *pre-post state matrix*: an approximately complete description of all allowed user decisions and their corresponding actions (resp. keystrokes or operators) changing the actual system state (pre-state) to the poststate; pre and poststates can, but must not be different. One special transition must be taken into account: the 'empty' user action. This means that in some states the system changes automatically to another state without any user input. To handle this aspect AMME takes not only user events into considerations but also system's output (e.g., output messages etc.). A state change can be detected (1) by user events or (2) by system reactions. Therefore an investigator can control the 'transition firing' process of the extracted Petri net either by user's operations or by system's output and hidden events, resp..

The Net Construction

A simple pattern matching algorithm looks for all 'elementary processes' in the logfile (and counts the frequency of each 'elementary process' for a further analysis of Markov-chains, see Schmid and Meseke 1991). A composition algorithm (the 'folding' operation) is now able to build up the Petri net combining all elementary processes by marking all detected states and transitions in the whole data structure which describes the complete system. For all systems with an infinite number of system states (e.g., in the process control area) the investigator has to choose such a low level of granularity that can be defined with a finite set of states.

Our analytical 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 (the whole state-space), on which the behaviour of individual processes is drawn. The task solving domain in our example is the whole dialog net structure of an interactive software program: the whole problem solving space. All parts of the user's keystroke sequence between two dialog states are elementary processes. All elementary processes can be combined to form a Petri net (the 'folding' operator). Each 'folded' Petri net is a formal description ('model') of the task solving structure of the users behaviour.

Generating and analysing logfiles in this automatic way enable the investigator to analyze enough samples to calculate applied statistics of large data sets. The next step is to find appropriate measures to describe different aspects of the generated Petri nets.

Analysing and Modelling Features Using AMME

Quantitative Aspects

Measuring Complexity

Measurable features of the task solving process are: task solving time (#TST), total number of states (#AS) and of transitions (#AT) used. These measurements can be easily done based on the analysis of the logfile itself. But, logfiles must be analysed with AMME to get the following two metrics: (1) number of different states (#DS) and (2) number of different transitions (#DT). These both numbers are the basis to calculate net complexity.

We investigated in (Rauterberg 1992a) the advantages and disadvantages of four different quantitative metrics in the context of an empirical investigation. With the Ccycle metric of McCabe (1976) we found a useful quantitative metric to measure complexity.

The complexity measured with Ccycle 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 to correct the result of Formula 1 in the case of a sequence (#T - #S = -1); the value of P in our context is one.

$$C_{\text{cycle}} = \#T - \#S + P \quad \text{with } \#S \leq \#T \text{ and } P=1 \quad (\text{Formula 1})$$

Ccycle of the example in the appendix A.4 is: $10 - 6 + 1 = 5$; the complexity of this net is five. But what could this number mean? McCabe (1976) interprets Ccycle as the *number of linear independent paths* through the net. Other interpretations of Ccycle are *number of holes* in a net or *number of alternative decisions* carried out by users. If #S is bigger than #T then we have to calculate Ccycle in a slightly different form (see also Rauterberg 1995). #F is the number of arrows in the net. C'cycle of the example in the appendix A.4 is: $20 - (10 + 6) + 1 = 5$, the same result as with Ccycle (this is only true for all nets with $\#S \leq \#T$).

$$C'_{\text{cycle}} = \#F - (\#T + \#S) + P \quad \text{with } \#S > \#T \text{ and } P=1 \quad (\text{Formula 2})$$

In the context of artificial intelligence we find a very common assumption (e.g., using protocol analysis techniques), that the complexity of the observable behaviour is *positively* correlated with the complexity of the underlying mental model. We assume, that this assumption is only valid for those cases, where the mental model is complete, and the observable task solving process is error free. But, these cases are only given for very simple systems!

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, p. 507). Transferring this broad definition to the man computer interaction could mean: the complexity of the user's mental model of the dialog system is given by the number of known dialog contexts (states) on one hand, and by the number of known dialog operations (transitions) on the other hand.

One important difference between novices and experts is the complexity of their mental models (Bainbridge 1991, p. 343). Novices have a simple task related knowledge structure, so they have to behave in a more heuristic manner to operate an interactive system. On the other hand, if the structure of the mental model of the experts is a more or less correct representation of the system structure, then they can behave and decide in a straight forward way to solve tasks.

In the investigation of Rauterberg (1992b) novices and experts were classified and selected by their amount of experiences with electronic data processing. The experience with electronic data processing was measured with an 115-item questionnaire and with structured interviews. The novice group (N=6) was instructed for 1.5 hours in handling the database system. The expert group (N=6) had 1,740 hours of experience in operating the same database system. Their total computer experience of about 7,500 hours was the result of their daily work using different types of computers and software systems. The duration of the actual task solving session was about 30 minutes. Each keystroke with a time stamp was recorded in a logfile. Each user needed about 50 minutes for the whole task solving process (4 tasks, individual sessions). The behavioural complexity (#BC) of each completed task solving process is measured as follows:

$$\#BC = C_{\text{cycle}}(\text{complete task solution}) \quad (\text{Formula 3})$$

A significant difference in #BC between novices and experts was found (N = 24, #BC_{nov} = 17 ± 6, #BC_{exp} = 12 ± 5; df=1, F = 10.3, p ≤ .003). This important result indicates, that the complexity of the observable behaviour correlates *negatively* with the complexity of the cognitive structure (the 'mental model'). The complexity of the necessary task knowledge can be either observed and measured with #BC or can be embedded in the cognitive structure. We explain this result as follows: If the cognitive structure is too simple, then the concrete task solving process must be filled up with a lot of heuristics or trial and error strategies. Learning how to solve a specific task with a given system mean, that #BC decreases and the complexity of the mental model increases (Rauterberg 1993, Rauterberg and Aeppli 1995).

Measuring 'Routinization'

A routine task can be identified by a large process description (= long logfile; e.g. #AT»10) and a small size of the embedded net structure (e.g. #DT≤10). The user is always running in loops and using the some system operations to solve the task; we designate this kind of tasks 'routine tasks'. The ratio of the total number of transitions in the process description (length of the logfile = #AT) to the number of different transitions in the folded net (#DT) is a good measure of the "degree of routinization" (#R).

$$\#R = \#AT / \#DT \quad (\text{Formula 4})$$

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.

Measuring 'Personality Styles'

To measure the dimension of 'action versus state orientation' all users filled out a personality questionnaire of Kuhl

(1981). This investigation is described in more detail in Rauterberg (1992b). The independent variables were (1) the level of expertise (novices versus experts) and (2) the four different tasks. All user actions were protocolled with time stamps in logfiles. With AMME we could extract all task dependent dialog states from 48 logfiles. The dependent variables are: (1) total task solving time (#TST) and (2) number of different states per Petri net (#DS). The ratio of #TST divided by #DS is the mean duration time per state. This average of the duration or dwell time per state can be interpreted as *user's thinking time* (#MTT) to plan the next action (see Formula 5).

$$\#MTT = \#TST / \#DS \quad (\text{Formula 5})$$

We aggregated all #MTT's over the four tasks and correlated this global value of #MTT per user with all scale scores of the 'action versus state orientation' questionnaire. We found a *negative* correlation between #MTT and scale-1: 'success leads to action orientation in thinking' (see Kuhl 1981). This significant correlation (R = -.75; p≤.005; N=12) means, that users with high scores in 'action orientation in thinking' caused by success have a short dwell time per state, and vice versa. Experts are--measured with the questionnaire--more 'action oriented' than novices (N=12, df=1, F=11.40, p=.007). We can conclude that state oriented persons need--on average--more time per dialog state, than action oriented persons.

Qualitative aspects

The 'Diagnostic' View

One valuable feature of a graphical net presentation is the fact, that one can see very quickly specific types of pattern (like the radiologist). One special pattern is the so called 'corona' pattern: many transitions around one state (see the transitions around the state s3 in the 'folded' net in Figure 1). Where does this corona pattern come from?

If the cognitive knowledge base of a user is not elaborated enough or the system output is inconsistent, then this user runs easily into an *interactive deadlock* situation. This type of situation is characterized by the fact, that the user does not know, how to leave the actual system state in an appropriate way. In these deadlock situations only trial and error behaviour helps--sometimes. This trial and error behaviour generates a *corona* pattern in the 'folded' Petri net.

One important feature of analysing logfiles with AMME is that the investigator has not to differentiate among *correct* and *incorrect* task solving actions beforehand: Any kind of task solving behaviour is welcome. The behavioural complexity of an interaction sequence with interactive deadlocks increases with the quota of trial and error strategies, that is all.

Analysing and Modelling of Task-Subtask Relationships

In problem solving and learning, in decision-making and prediction it is important that people check their own performance against the outcome obtained. They will be able to learn from their experiences provided they have feedback about a reason for the outcome of their decisions (Annett

1972). They may be able to concentrate on their 'weak spots'. If we use the 'planning time' or 'dwell time per dialog state' as a criterion to trigger subnet construction (Ackermann 1994, pp. 190ff), then we will get another type of net: the *action plans* of the task solving process.

First, we have to calculate the average over all dwell times per state. The next step is the construction of all subnets--as representatives of subtasks--triggered by this criterion. Then, these subnets can be separately analysed with AMME. To go this way, we have only to update the state definition list in the system description file: the first state of each subtask has to be put at the beginning of the definition list. So, we will get normally for each subtask a new, corresponding system description file.

Analysing and Modelling of Learning Processes

Modelling a learning process we use finite place transition nets with unmarked tokens (see Rauterberg 1995). Using a timed Petri-net simulator (e.g., PACE) we are able to model time aspects in a very simple way. With a simulator as PACE we can analyze the extracted and supplemented models in a discrete or stochastic fashion. The task solving process of expert user can be supplemented with a simple model of long-term memory. The activation probability of each action in a state, that was unsuccessful, decrease after one, two or three unsuccessful executions to zero. We model this aspect adding special S-elements, that have one, two or three unmarked tokens. All successful actions get an S-element, that preserve the activation probability of the transition. If we try to model a structure of different plans, then we need a more elaborated extension of the Petri net.

The similarity in KNOT is determined by the correspondence of transitions in two nets. Two identical nets will yield a similarity of one and two complementary nets will yield similarity of zero. This measure is roughly the proportion of all the transitions in either net that are in both nets. With this measure of similarity we can compare the net at a given learning stage against a reference net with the decision structure of the knowledge, which have to be learned by the user.

Conclusions

If a user interacts with a system, that can be described with a finite discrete state transition net, then his behaviour can be traced with a sequence of states and transitions ($s^i \rightarrow [t^i] \rightarrow (s^{i+1}) \rightarrow [t^{i+1}] \rightarrow (s^{i+2}) \rightarrow [t^{i+2}] \rightarrow \dots$). Many different systems controlled by users can be described with finite state transition nets. To investigate the interaction processes with these kinds of reasonable complex systems, we need support of special tools. We present a tool kit, that analyze incomplete sequences of states and transitions, to come up with the underlying net structure and different measures of complexity of the decision process described by the state transition sequence.

On the basis of the empirical result, that the complexity of the observable behaviour of novices is significantly larger than the complexity of experts' task solving processes, we can conclude, that the complexity of the behaviour is *negatively* correlated with the cognitive complexity of the corre-

sponding mental model. So, we are able to estimate the cognitive complexity based on the measurement of the observed behaviour.

One of the most interesting aspect of nets constructed with our analysing tool is the possibility to measure the behavioural and decision effort in a simple fashion. Now we are able to investigate the explicit *and* implicit learning process of users in handling an interactive system.

To measure complexity of a system described with a state transition matrix in a quantitative way is one side; the other side is to transform the structure of a given system in an 'appropriate form'. One qualitative approach to figure complexity is drawing the 'net structure' of the system (Jones 1993). If we use Petri nets instead of the equivalent state transition formalism (Wasserman 1985), we can simulate the user's mental model in an executable form with Petri net simulators, too (Rauterberg 1995).

The possibility to detect an interactive deadlock is another important feature and of great interest for a system designer. As we mentioned above an interaction process recorded in a logfile can be folded to the underlying net structure, that can be scanned for typical pattern. As a 'radiologist' it is now possible to produce a couple of pictures of decision processes and scan them for interesting pattern.

Acknowledgements

We gratefully acknowledge the invaluable support in developing the analysing tool by Jens Hofmann, Claude Leuchter, Jack Rudnik and Martin Roth. The preparation of this paper was supported by the BMFT (AuT program) grant number 01 HK 706-0 as part of the BOSS 'User oriented Software Development and Interface Design' research project and the ADI Software Inc. in Karlsruhe (Germany).

References

- Ackermann, D. (1984) Handlungsspielraum, mentale Repräsentation und Handlungsregulation am Beispiel der Mensch-Computer Interaktion (Dissertation, Work and Organization Psychology Unit, ETH Zuerich).
- AMME (1996) *Download address*: URL: <http://www.ifap.bepr.ethz.ch/~rauter/amme.html>
- Alty, J.L. (1984) The application of path algebras to interactive dialogue design, *Behaviour and Information Technology*, 3, 119-132.
- Annett, J. (1972) *Feedback and human behaviour* (Penguin).
- 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.
- Booth, P.A. (1991) Errors and theory in human-computer interaction, *Acta Psychologica*, 78, 69-96.
- Card, S.K., Moran, T.P. and Newell, A. (1983) *The psychology of human computer interaction*. (Erlbaum).
- Churchill, E. (1992) The formation of mental models: are 'device instructions' the source?, pp. 3-16. In: G.C. van der Veer, M.J. Tauber, S. Bagnara and A. Antalovits,

- Eds., Human-Computer Interaction: Tasks and Organisation (CUD, Roma).
- de Haan, G., van der Veer, G.C. and van Vliet, J.C. (1991) Formal modelling techniques in human-computer interaction, *Acta Psychologica*, 78, 27-67.
- Dolk, D.R. (1993) An introduction to model integration and integrated modeling environments, *Decision Support Systems*, 10, 249-254.
- Ericsson, K.A. and Simon, H.A. (1984) Protocol analysis (MIT Press).
- Geoffrion, A.M. (1989) Integrated modeling systems, *Computer Science in Economics and Management*, 2, 3-15.
- Genrich, H.J., Lautenbach, K. and Thiagarajan, P. S. (1980) Elements of general net theory, pp. 21-163. In: W. Bauer, Ed., *Lecture Notes in Computer Science 84 'Net Theory and Applications'* (Springer).
- 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).
- Johnson, S. C. (1975) YACC – yet another compiler-compiler, Technical Report No. 32 (Bell Laboratories).
- Jones, C. (1993) An integrated modeling environment based on attributed graphs and graph-grammars. *Decision Support Systems*, 10, 255-275.
- Kellog, W.A. and Breen, T.J. (1990) Using Pathfinder to evaluate user and system models, pp. 179-195. In: R. W. Schvaneveldt, Ed., *Pathfinder Associative Networks: Studies in Knowledge Organization* (Ablex).
- 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.
- Kuhl, J. (1981) Motivational and functional helplessness: the moderating effect of state vs. action orientation, *Journal of Personality and Social Psychology* 40, 155-170.
- McCabe, T. (1976) A complexity measure, *IEEE Transactions on Software Engineering*, SE-2, 308-320.
- McDaniel, E. and Lawrence, C. (1990) Levels of cognitive complexity: an approach to the measurement of thinking (Springer).
- Mössenböck, H. (1990) Coco/R – a generator for fast compiler front-ends, Technical Report No. 127 (Computer Science Department of the ETH Zuerich).
- Oberquelle, H. (1984) On models and modeling in human-computer co-operation, pp. 26-45. In: G.C. van der Veer, M. J. Tauber, T.R.G. Green and P. Gorny, Eds., *Lecture Notes in Computer Science 178 'Readings on Cognitive Ergonomics'* (Springer).
- Oberquelle, H., Kupka, I. and Maass, S. (1983) A view of human-machine communication and co-operation, *International Journal of Man-Machine Studies*, 19, 309-333.
- PACE (1995) *Distributor address*: GPP Gesellschaft für Prozessrechnerprogrammierung, Kolpingring 18a, D-8024 Oberhaching bei München, Germany.
- Pervin, L.A. (1984) *Personality* (Wiley).
- Petri, C.A. (1980) Introduction to general net theory, pp. 1-19. In: W. Bauer, Ed., *Lecture Notes in Computer Science 'Net Theory and Applications'* (Springer).
- Peterson, J.L. (1981) *Petri net theory and the modeling of systems* (Englewood Cliffs).
- Rauterberg, M. (1992a) A method of a quantitative measurement of cognitive complexity, pp. 295-307. In: G.C. van der Veer, M.J. Tauber, S. Bagnara and A. Antalovits, Eds., *Human-Computer Interaction: Tasks and Organisation* (CUD, Roma).
- Rauterberg, M. (1992b) 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. (1993) AMME: an automatic mental model evaluation to analyze user behaviour traced in a finite, discrete state space, *Ergonomics*, 36, 1369-1380.
- Rauterberg, M. (1995) From novice to expert decision behaviour: a qualitative modelling approach with Petri nets, pp. 449-454. In: Y. Anzai, K. Ogawa and H. Mori, Eds., *Symbiosis of Human and Artifact: Human and Social Aspects of Human-Computer Interaction*. (Vol. 2).
- Rauterberg, M. and Aeppli, R. (1995) Learning in man-machine systems: the measurement of behavioural and cognitive complexity, pp. 4685-4690. In: *Proceedings of IEEE International Conference on Systems, Man and Cybernetics 'Intelligent Systems for the 21st Century'*. (Vol. 5, IEEE Catalog No. 95CH3576-7).
- Reisig, W. (1992) *A Primer in Petri Net Design* (Springer).
- Reisner, P. (1981) Formal grammar and human factors design of an interactive graphics system, *IEEE Transactions on Software Engineering*, SE-7, 229-240.
- Reisner, P. (1984) Formal grammar as a tool for analyzing ease of use, pp. 53-78. In: J.C. Thomas and M.L. Schneider, Eds., *Human Factors in Computing Systems* (Ablex).
- 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.
- Schmid, U. and Meseke, B. (1991) Deskription und Analyse komplexer Verhaltenssequenzen: Benutzerstrategien beim Arbeiten mit CAD-Systemen, *Zeitschrift für experimentelle und angewandte Psychologie*, 38, 307-320.
- Schröder, O., Frank, K.D., Kohnert, K., Möbus, C. and Rauterberg, M. (1990) Instruction-based knowledge acquisition and modification: the operational knowledge for a functional, visual programming language, *Computers in Human Behaviour*, 6, 31-49.
- Schvaneveldt, R.W. (1990, Ed.) *Pathfinder associative networks - studies in knowledge organization* (Ablex).
- Scott, W.A., Osgood, D.W. and Peterson, C. (1979) Cognitive structure: theory and measurement of individual differences (Wiley).
- Törn, A.A. (1985) Simulation nets, a simulation modeling and validation tool, *Simulation*, 45, 71-75.
- Wasserman, A.I. (1985) Extending state transition diagrams for the specification of human-computer interaction, *IEEE Transactions on Software Engineering*, SE-11, 699-713.