AMME: A Petri Net Based Analyzing and Modelling Tool Kit for Logfiles in Human-Computer Interaction

Matthias Rauterberg

Work- and Organization Psychology Unit
Swiss Federal Institute of Technology (ETH)
Nelkenstr. 11, CH-8092 Zurich, Switzerland
Tel: +41-1-632 7082, Email: rauterberg@ifap.bepr.ethz.ch

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 behavior 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 analyzing tool kit can handle the recorded decision and action sequences and come 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 behavior.

Keywords: Petri net modeling, human-computer interaction, logfile analysis

1. Introduction

Several methods are developed and used to study cognition's: questionnaires [46], protocol analysis [11], formal models [21], scoring rationale of observable behavior ('interpretive exercise' [24]). "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" ([14], p.301). The method presented in this paper was developed to support the 'cognitive scientist' in modeling human decision behavior.

The normal design cycle to construct a formal model of human behavior is a task dependent top down approach. Given a top down constructed model, it has to be validated with empirical data (e.g., [44]). In this paper we present an alternative automatic bottom up approach to construct a formal description of user decision behavior. 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 [41] 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 [46] or other scoring rationale techniques [24]. The presented approach is an integrated modeling environment based on Petri nets (see [19]).

1.1. On models and modeling in human-computer interaction

Geoffrion [12] describes modeling processes on four different levels: modeling traditions, modeling paradigms, model classes, and specific models. Our modeling tradition is human-computer interaction and cognitive ergonomics. All classes of mental and/or task models describing user behavior are our *modeling 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. [37]).

Dolk [10] differentiates between three main schools of model representation: structured modeling, logic modeling, and graph grammars. Our approach can be subsumed under graph grammars. There are different formalisms for constructing models of cognitive processes: MAD [42], Task Action Grammar [30], Command Language Grammar [25], Cognitive Complexity Theory [21], GOMS [7], and different kinds of grammars BNF [39], EBNF [40], UAN [15] 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. Then he can try to prove his model with 'error free' data (e.g., [8]). This is often difficult, time consuming, and expensive [20]. 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 [27] shows, that several notions of 'model' are in use: (1) a model of an axiom system, (2) a known system with structure and behavior 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., [44]). Class (4) models are useful to classify and explain phenomena of the modeled 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 symbols, 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 behavior). 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.

1.2. Modeling 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) [33]. The flow relation links only different node types and leaves no node isolated [32]. 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 [47] 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 modeling (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)* [28].

Palanque, Bastide and Senges [29] proposed a straight forward way for translating from a UAN description [15] to a Petri net. This approach provides three advantages: (1) "The ambiguities of the task models are solved during the translation process." (2) "The use of Petri nets for modeling the interactive application, cooperation betweeen the model of the task and the model of the system can be mathematically checked, …" (3) "As the model of the system is embedded at run time, it can be used to provide contextual help about the behavior of the system, …" ([29], p. 494).

Bauman and Turano [5] showed, that Petri nets are equivalent to formalism based on production rules (like CCT of Kieras and Polson [21]). In this sense, the presented approach can be subsumed under 'logic modeling', too.

Applying the path algebra approach of Alty [2] 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" ([2], p. 125).

2. The method

2.1. The basic idea

The main operations (relations) between two Petri nets are abstraction, embedding and folding [13].

2.1.1. 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. 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.

2.1.2. Complete versus incomplete task solving descriptions

If the observable behavior 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 behavior 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 difficulty. 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 prize we have to pay is the definition of a complete state transition description of all relevant transitions beforehand.

2.2. The Petri net construction process

2.2.1. The architecture of the analysing tool kit

The whole system of our analyzing tool kit consists of five different programs:

(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 (see chapter 2.2.4). A logfile can also be hand written by the investigator (e.g., based on protocols of observations).

(B) The *net generation program* AMME [16] 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 protocoll 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; (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 analyzing software SEQUENZ, and (5) a PostScript file ("*.ps") to print the net graphic for pattern matching 'by hand'.

(C) The *Petri net simulator* PACE [9] is commercially available; 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 analyzing program* KNOT [17] [45] we can compute the similarity between pairs of nets. 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 analyzing software SEQUENZ [43] 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.

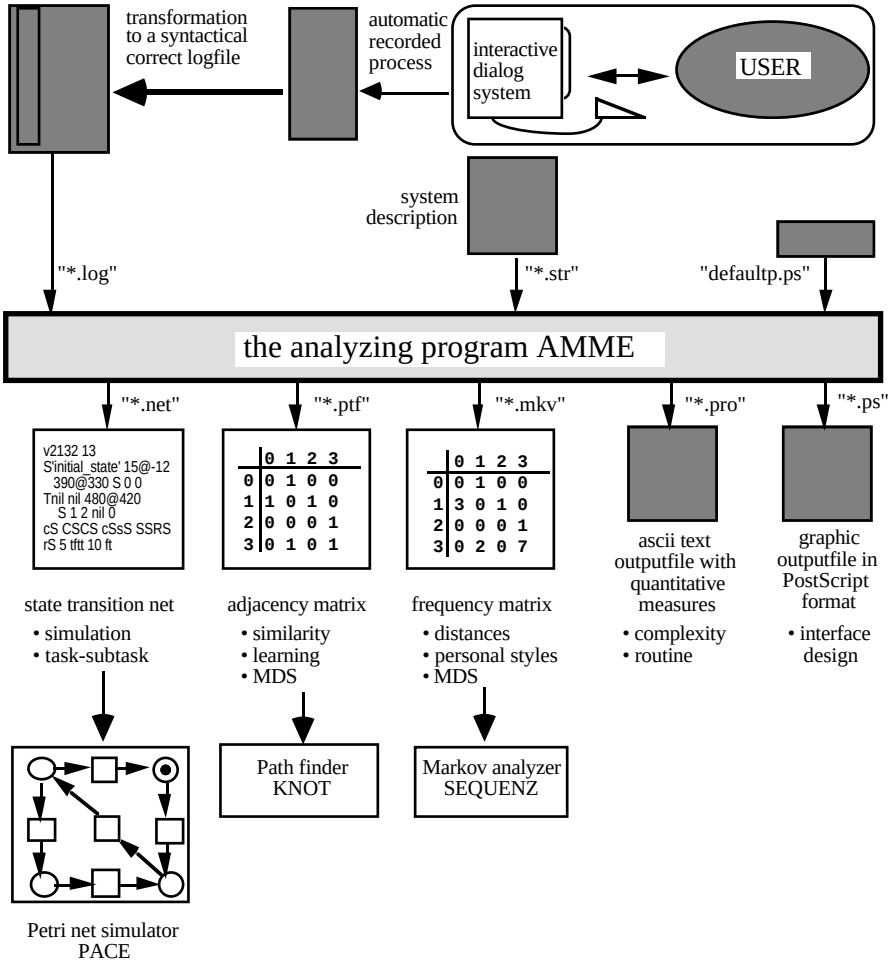


Figure 1: An overview over all involved programs and files of the analyzing tool kit.

The current version of AMME is restricted to process descriptions that can be traced in a *finite, discret 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 [26], YACC [18], 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).

2.2.2. The data and program structure of AMME

We choose an open data structure to maximize the flexibility of AMME (see Figure 2). The only restriction is the number of dialog states (at the moment to a maximum of 180 definable state identifiers). Each identifier can be an alpha-numeric name up to 60 characters. All defined transitions are organized in an open list (the limitation for this list depends only on the actual amount of main memory). The 'modifier' and 'propagator' lists are used for future extensions to make transitions context-sensitive (definable execution conditions). This extension seems to be necessary to increase to applicability of AMME to interactive systems that can not be described with a complete state-transition matrix beforehand.

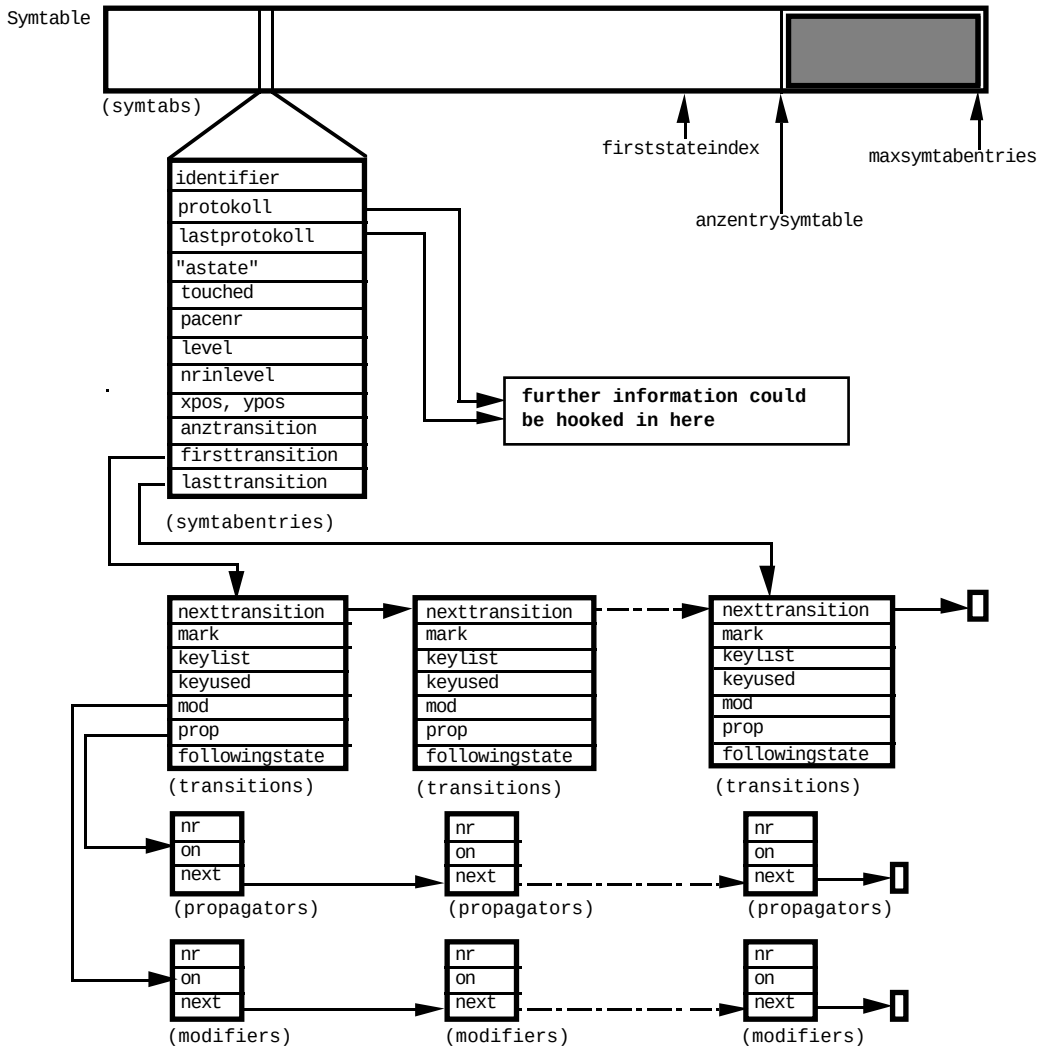


Figure 2: The data structure of AMME.

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.

2.2.3. The user's logfile

As shown in Figure 1 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_MENUE", "LOG_MESSAGE", "LOG_ERROR", "LOG_HELP", "LOG_USRTIME", and "LOG_SYSTIME" (see Table 1 and the example in the appendix A.1 and A.2).

Table 1: The different logfile eventy types and their correspondents in the system description.

logfile events	*content of the system description file*	*entries in *.log and*	*in *.str*
LOG_KEYBD : α	[α] ;# with α ∈ ASCII[032…127]	LOG_KEYBD : x	[x]
LOG_KEYBD :	[BL] ;# Blank,space	LOG_KEYBD :	[BL]
LOG_KEYBD : α	[ALL] ;# any α	LOG_KEYBD : r	[ALL]
LOG_KEYBD : <F β>	[F_β] ;# function key no.	LOG_KEYBD : <F_10>	[F_10]
LOG_KEYBD : <S 13>	[S_13] ;# ENTER, Return	LOG_KEYBD : <S_13>	[S_13]
LOG_MENUE : β	[M_β] ;# menue-no., β ∈ {0…99}	LOG_MENUE : 3	[M_3]
LOG_MESSAGE: β	[G_β] ;# message-no.	LOG_MESSAGE: 18	[G_18]
LOG_ERROR : β	[E_β] ;# error-no.	LOG_ERROR : 99	[E_99]
LOG_HELP : β	[H_β] ;# help-no.	LOG_HELP : 21	[H_21]
LOG_USRTIME: γ	# with γ ∈ {0,0…999,0} time unit	LOG_USRTIME: 11,0	nothing
LOG_SYSTIME: γ	# with γ ∈ {0,0…999,0} time unit	LOG_SYSTIME: 126,0	nothing

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 they are summed up separately during the parsing process.

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

2.2.4. 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 behavior 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 lowest possible level is each keypress or mouse click itself. This level is only of interest if one try to model user behavior on the keystroke level (see KLM in [7]).

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" (see Table 2).

Table 2: The syntax definition for the system description file (filename with extension ".str").

```
SM =              states {comments} transactions {comments} "END".
states =          "STATES" "=" statename {"," statename} ";".
transactions =    "TRANSITIONS" "=" {prestatename "=>" poststatename ";"}
                  "|".
prestatename =    statename.
poststatename =   statename "[" transitionname {"," transitionname} "]"
                  ";".
statename =       char {char | digit | "."}.
transitionname =  char | digit | "BL" | symbolchar "_" {number} | "ALL".
symbolchar =      "F" | "S" | "M" | "G" | "E" | "H".
comments =        "#" {" " | char | number} "CR".
number =          digit | digit digit.
char =            "a".."Z".
digit =           "0".."9".
```

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 (prestate) 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 (messages etc.). A state change can be detected (1) by user events or (2) by system reactions. Therefor an investigator can control the 'transition firing' process of the extracted Petri net either by user's operations or by system's output (see the example in the appendix A.3 and A.4).

2.2.5. 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 Marcov-chains, see [43]). 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 data structure (see Figure 2).

This 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 state-space), on which the behavior 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 behavior.

Generating and analyzing 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.

3. Analyzing and modelling features using AMME

3.1. Quantitative aspects

3.1.1. 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 [34] the advantages and disadvantages of four different quantitative metrics in the context of an empirical investigation. With the C_{cycle} metric of McCabe [23] we found a useful quantitative metric to measure complexity.

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 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.

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

C_{cycle} 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 [23] interprets C_{cycle} as the *number of linear independent paths* through the net. Other interpretations of C_{cycle} 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 C_{cycle} in a slightly different form (see also [37]). #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 C_{cycle} (this is only true for all nets with #S ≤ #T).

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

In the context of artificial intelligence we find a very common assumption (e.g., using protocol analysis techniques), that the complexity of the observable behavior 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)" ([31], 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 ([4], 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 [35] 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 behavioral complexity (#BC) of each completed task solving process is measured as follows:

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

A significant difference in #BC between novices and experts was found (N = 24, $\#BC_{nov} = 17 \pm 6$, $\#BC_{exp} = 12 \pm 5$; df=1, F = 10.3, $p \leq .003$). This important result indicates, that the complexity of the observable behavior 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 [see 29b].

3.1.2. 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).

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

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.

3.1.3. Measuring 'personality styles'

To measure the dimension of 'action versus state orientation' all users filled out a personality questionnaire of Kuhl [22] (the same investigation as described in chapter 3.1.1 and [35]). 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).

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

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 [22]). This significant correlation ($R = -.75$; $p \leq .005$; $N=12$) 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 a greater duration per dialog state, than action oriented persons.

3.2. Qualitative aspects

3.2.1. 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 Figure 2 in [36] on p. 1373). 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 dialog state in an appropriate way. In these deadlock situations only trail and error behavior helps--sometimes. This trial and error behavior generates a *corona* pattern in an extracted 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 behavior is wellcome. The behavioral complexity of an interaction sequence with interactive deadlocks increases with the quota of trial and error strategies, that is all.

3.2.2. Analyzing and modeling 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 [3]. 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 ([1], 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 analyzed 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 at the beginning of the definition list. So, we will get normally for each subtask a new system description file.

3.2.3. Analyzing and modeling of learning processes

Modeling a learning process we use finite place transition nets with unmarked tokens. Using the Petri-net simulator PACE we are able to model time aspects in a very simple way. As each simulator, PACE offers the possibility of simulation, so we can analyze the extracted and supplemented models in a discrete or stochastic fashion (cf. [38]). 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 plany, 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. The 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.

4. Conclusions

If a user interacts with a system, that can be described with a finite discrete state transition net, then his behavior can be traced with a sequence of states and transitions (s') -> [t'] -> (s'') -> [t''] -> (s') -> [t'] -> (s'') -> [t''] -> ... . 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 behavior of novices is significantly larger than the complexity of experts' task solving processes, we can conclude, that the complexity of the behavior is *negatively* correlated with the cognitive complexity of the corresponding mental model. So, we are able to estimate the cognitive complexity based on the measurement of the observed behavior.

One of the most interesting aspect of nets constructed with our analyzing tool is the possibility to measure the behavioral and decision effort in a simple fashion. Now each researcher is able to investigate the 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 [19]. If we use Petri nets instead of the equivalent state transition formalism [48], we can simulate the user's mental model in an executable form with Petri net simulators, too.

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 log-file 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.

Acknowledgments

We gratefully acknowledge the invaluable support in developing the analyzing 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

[1] D. Ackermann, Handlungsspielraum, mentale Repräsentation und Handlungsregulation am Beispiel der Mensch-Computer Interaktion (Dissertation, Work and Organizational Psychology Unit, Swiss Federal Institute of Technology, 1984).

[2] J.L. Alty, The application of path algebras to interactive dialogue design, Behaviour and Information Technology, 3 (1984) 119-132.

[3] J. Annett, Feedback and human behavior (Penguin, 1972).

[4] L. Bainbridge, The 'cognitive' in cognitive ergonomics, Le Travail humain, 54((1991) 337-343.

[5] R. Bauman and T.A. Turano, Production based language simulation of Petri nets, Simulation, 47 (1986) 191-198.

[6] C. Cachin and A. Plimpton, Zusammenhang zwischen Blickbewegung und Mauscursorbewegungen, unpublished Technical Report (Work and Organizational Psychology Unit, Swiss Federal Institute of Technology, Zuerich, 1992).

[7] S.K. Card, T.P. Moran and A. Newell, The psychology of human computer interaction. (Erlbaum 1983).

[8] E. Churchill, The formation of mental models: are 'device instructions' the source?. Human-Computer Interaction: Tasks and Organisation (G. C. van der Veer, M. Tauber, S. Bagnara and M. Antalovits, eds.) CUD (Roma, Italy), 3-16

[9] J. Dähler, The Petri net simulator PACE, (Grossenbacher Electronic Inc., Spinnereistrasse 8, CH-9008 St. Gallen, Switzerland, 1989)

[10] D.R. Dolk, An introduction to model integration and integrated modeling environments Decision Support Systems, 10 (1993) 249-254.

[11] K.A. Ericsson and H.A. Simon, Protocol analysis (MIT Press, 1984).

[12] A.M. Geoffrion, Integrated modeling systems, Computer Scinece in Economics and Management, 2 (1989) 3-15.

[13] H.J. Genrich, K. Lautenbach and P. S. Thiagarajan, Elements of general net theory, pp. 21-163. In: W. Bauer, Ed., Lecture Notes in Computer Science 84 'Net Theory and Applications' (Springer, 1980).

[14] T.R.G. Green and J.M. Hoc, What is cognitive ergonomics?. Le Travail humain, 54 (1991) 291-304.

[15] D. Hix and R. Hartson, Developing user interfaces (Wiley, 1993).

[16] J. Hofmann and J. Rudnik, PACEGEN: ein automatisches Logfile-Auswertungsprogramm zur Generierung von Petri Netzen, Unpublished Technical Report (Work and Organization Psychology Unit, Swiss Federal Institute of Technology, Zuerich, 1991).

[17] Interlink, Inc., The KNOT software (P.O. Box 4086 UPB, Las Cruces, NM 88003, USA, 1991).

[18] S. C. Johnson, YACC – yet another complier-compiler, Technical Report No. 32 (Bell Laboratories, 1975).

[19] C. Jones, An integrated modeling environment based on attributed graphs and graph-grammers. Decision Support Systems, 10 (1993) 255-275.

[20] W.A. Kellog and T.J. Breen, Using Pathfinder to evaluate user and system models, pp. 179-195. In: R. W. Schvaneveldt, Ed., Pathfinder Associative Networks: Studies in Knowledge Organization (Ablex, 1990).

[21] D.E. Kieras and P.G. Polson, An approach to the formal analysis of user complexity, International Journal of Man-Machine Studies, 22 (1985) 365-394.

[22] J. Kuhl, Motivational and functional helplessness: the moderating effect of state vs. action orientation. Journal of Personality and Social Psychology 40 (1981) 155-170.

[23] T. McCabe, A complexity measure, IEEE Transactions on Software Engineering, SE-2 (1976) 308-320.

[24] E. McDaniel and C. Lawrence, Levels of cognitive complexity: an approach to the measurement of thinking (Springer, 1990).

[25] T.P. Moran, The command language grammar: a representation for the user interface of interactive computer systems, International Journal of Man-Machine Studies, 15 (1981) 3-50.

[26] H. Mössenböck, Coco/R – a generator for fast compiler front-ends, Technical Report No. 127 (Computer Science Department of the ETH, 1990).

[27] H. Oberquelle, 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 - Mind and Computers' (Springer, 1984).

[28] H. Oberquelle, I. Kupka and S. Mass, A view of human-machine communication and cooperation, International Journal of Man-Machine Studies, 19 (1983) 309-333.

[29] P. Palanque, R. Bastide and V. Senges, Task model–system model: towards an unifying formalism, pp. 489-494. In: Y. Anzai, K. Ogawa and H. Mori, Eds., Symbiosis of Human and Artifact: Human and Social Aspects of Human-Computer Interaction (Vol. 2, Elsevier, 1995).

[30] S.J. Payne and T.R.G. Green, Task-action grammars: a model of the mental representation of task languages, Human Computer Interaction, 2 (1986) 93-133.

[31] L.A. Pervin, Personality (Wiley, 1984).

[32] C.A. Petri, Introduction to general net theory, pp. 1-19. In: W. Bauer, Ed., Lecture Notes in Computer Science 'Net Theory and Applications' (Springer, 1980).

[33] J.L. Peterson, Petri net theory and the modeling of systems (Englewood Cliffs, 1981).

[34] M. Rauterberg, 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 1992).

[35] M. Rauterberg, An empirical comparison of menu-selection (CUI) and desktop (GUI) computer programs carried out by beginners and experts, Behaviour & Information Technology, 11 (1992) 227-236.

[36] M. Rauterberg, AMME: an automatic mental model evaluation to analyze user behaviour traced in a finite, discrete state space, Ergonomics 36 (1993) 1369-1380.

[37] M. Rauterberg, 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, Elsevier, 1995).

[38] M. Rauterberg and R. Aeppli, 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,Piscataway, 1995).

[39] P. Reisner, Formal grammar and human factors design of an interactive graphics system, IEEE Transactions on Software Engineering, SE-7 (1981) 229-240.

[40] P. Reisner, 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, 1984).

[41] P.M. Sanderson, A.G. Verhage and R.B. Fuld, State-space and verbal protocol methods for studying the human operator in process control, Ergonomics, 32 (1989) 1343-1372.

[42] D.L. Scapin and C. Pierret-Golbreich, Towards a method for task description: MAD, pp. 371-380. In: L. Berlinguet and D. Berthelette, Eds., Work with Display Units 89 (Elsevier, 1990).

[43] U. Schmid and B. Meseke, Deskription und Analyse komplexer Verhaltenssequenzen: Benutzerstrategien beim Arbeiten mit CAD-Systemen, Zeitschrift für experimentelle und angewandte Psychologie, 38 (1991) 307-320.

[44] O. Schröder, K.D. Frank, K. Kohnert, C. Möbus and M. Rauterberg, Instruction-based knowledge acquisition and modification: the operational knowledge for a functional, visual programming language, Computers in Human Behavior, 6 (1990) 31-49.

[45] R.W. Schvaneveldt, Ed., Pathfinder associative networks - studies in knowledge organization (Ablex, 1990).

[46] W.A. Scott, D.W. Osgood and C. Peterson, Cognitive structure: theory and measurement of individual differences (Wiley, 1979).

[47] A.A. Törn, Simulation nets, a simulation modeling and validation tool, Simulation, 45 (1985) 71-75.

[48] A.I. Wasserman, Extending state transition diagrams for the specification of human-computer interaction, IEEE Transactions on Software Engineering, SE-11 (1985) 699-713.

Appendix

A.1. The appearence of a concrete "login" and "logout" procedure on the screen of a Unix server:

output and input on the screen during each step	content of the logfile
initial state	LOG_KEYBD : <F 10>
UNIX(r) System V Release 4.0 (marshall)	LOG_MESSAGE: 1
login: rauterberg	LOG_KEYBD : r...g
Password:	LOG_KEYBD : *...*
Login incorrect	LOG_MESSAGE: 2
login: rauter	LOG_KEYBD : r...r
Password:	LOG_KEYBD : *...*
Last login: Wed Feb 7 19:16:57 from rota.ethz.ch	LOG_MENUE : 3
Sun Microsystems Inc. SunOS 5.4 Generic July 1994	LOG_MESSAGE: 4
Wed Feb 7 19:17:54 MET 1996	LOG_MESSAGE: 5
You have no mail.	LOG_MESSAGE: 24
marshall:/export/home/rauterberg!51> ls -l	LOG_KEYBD : l...l
total 2	LOG_MENUE : 16
drwxr-xr-x 7 rauter ifap 512 Feb 1 20:28 mac/	LOG_MENUE : 17
marshall:/export/home/rauterberg!52> x	LOG_KEYBD : x
initial state	

[with the alias "x" := "logout"]

A.2. The complete content of the user's logfile that describes the "login" and "logout" task:

```
LOG_KEYBD  : <F 10>
LOG_USRTIME: 1,0
LOG_MESSAGE: 1
LOG_KEYBD  : r
LOG_USRTIME: 1,0
LOG_KEYBD  : a
LOG_USRTIME: 1,0
LOG_KEYBD  : u
LOG_USRTIME: 1,0
LOG_KEYBD  : t
LOG_USRTIME: 1,0
LOG_KEYBD  : e
LOG_USRTIME: 1,0
LOG_KEYBD  : r
LOG_USRTIME: 1,0
LOG_KEYBD  : b
LOG_USRTIME: 1,0
LOG_KEYBD  : e
LOG_USRTIME: 1,0
LOG_KEYBD  : r
LOG_USRTIME: 1,0
LOG_KEYBD  : g
LOG_USRTIME: 1,0
LOG_KEYBD  : <S_13>
LOG_USRTIME: 1,0
LOG_SYSTIME: 9,0
LOG_KEYBD  : *
LOG_USRTIME: 1,0
LOG_KEYBD  : *
LOG_USRTIME: 1,0
LOG_KEYBD  : *
LOG_USRTIME: 1,0
LOG_KEYBD  : *
LOG_USRTIME: 1,0
```

```
LOG_KEYBD  : *
LOG_USRTIME: 1,0
LOG_KEYBD  : *
LOG_USRTIME: 1,0
LOG_KEYBD  : <S_13>
LOG_USRTIME: 1,0
LOG_SYSTIME: 49,0
LOG_MESSAGE: 2
LOG_SYSTIME: 8,0
LOG_KEYBD  : r
LOG_USRTIME: 1,0
LOG_KEYBD  : a
LOG_USRTIME: 1,0
LOG_KEYBD  : u
LOG_USRTIME: 1,0
LOG_KEYBD  : t
LOG_USRTIME: 1,0
LOG_KEYBD  : e
LOG_USRTIME: 1,0
LOG_KEYBD  : r
LOG_USRTIME: 1,0
LOG_KEYBD  : <S_13>
LOG_USRTIME: 1,0
LOG_SYSTIME: 9,0
LOG_KEYBD  : *
LOG_USRTIME: 1,0
LOG_KEYBD  : *
LOG_USRTIME: 1,0
LOG_KEYBD  : *
LOG_USRTIME: 1,0
LOG_KEYBD  : *
LOG_USRTIME: 1,0
```

```
LOG_KEYBD  : *
LOG_USRTIME: 1,0
LOG_KEYBD  : <S_13>
LOG_USRTIME: 1,0
LOG_SYSTIME: 24,0
LOG_MENUE  : 3
LOG_SYSTIME: 2,0
LOG_MESSAGE: 4
LOG_SYSTIME: 7,0
LOG_MESSAGE: 5
LOG_MESSAGE: 24
LOG_SYSTIME: 41,0
LOG_KEYBD  : l
LOG_USRTIME: 1,0
LOG_KEYBD  : s
LOG_USRTIME: 1,0
LOG_KEYBD  :
LOG_USRTIME: 1,0
LOG_KEYBD  : -
LOG_USRTIME: 1,0
LOG_KEYBD  : l
LOG_USRTIME: 1,0
LOG_KEYBD  : <S_13>
LOG_USRTIME: 1,0
LOG_SYSTIME: 17,0
LOG_MENUE  : 16
LOG_MENUE  : 17
LOG_SYSTIME: 7,0
LOG_KEYBD  : x
LOG_USRTIME: 1,0
LOG_KEYBD  : <S_13>
LOG_USRTIME: 1,0
```

A.3. The content of a struture file that describes the system with all relevant states for the analysed tasks:

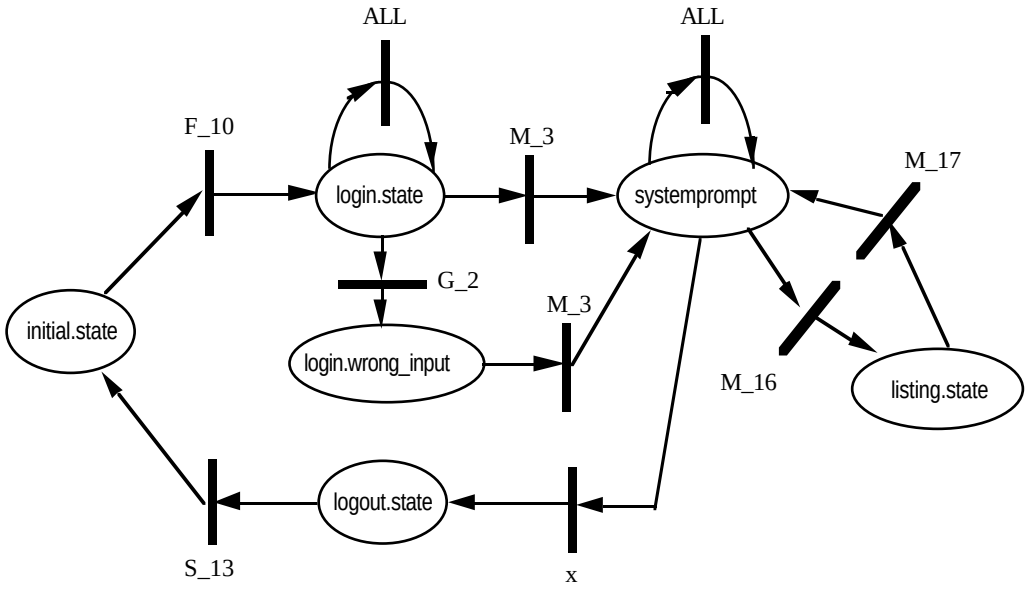
```
# list of all relevant dialog states of the interactive system

STATES  =

# states for correct input behavior
# [Note: the first state in this list defines the starting state for the analysis. One
#        attractive consequence is that any part of a logfile can be analysed through
#        updating this state descriptor list]
     initial.state ,
     login.state,
     systemprompt,
     listing.state,
     logout.state,

# states for incorrect input behavior

     login.wrong_input,
     ;

# list of all known transisitions
TRANSITIONS =
    initial.state     =>  login.state       [F_10] ;
    login.state       =>  systemprompt      [M_3] ;
    login.state       =>  login.wrong_input [G_2] ;
    login.state       =>  login.state       [ALL] ;
    listing.state     =>  systemprompt      [M_3] ;
    login.wrong_input =>  systemprompt      [M_3] ;
    systemprompt      =>  logout.state      [x] ;
    systemprompt      =>  listing.state     [M_16, M_17] ;
    systemprompt      =>  systemprompt      [G_4, G_5, G_24, ALL] ;
    logout.state      =>  initial.state     [S_13] ;
    |
END
# [Note: the operator ALL is very powerful to eliminate all unnecessary keystroke
#        events; before define a transition with this operator make sure that there
#        is a another correct transition to leave the state beforehand]
```

A.4. The generated Petri net for the given example: