An analysing and modelling tool kit for human-computer interaction

An Analysing and Modelling Tool Kit for Human-Computer Interaction

M. Rauterberg & M. Fjeld

Institute for Hygiene and Applied Physiology (IHA)
Swiss Federal Institute of Technology (ETH)
Clausiusstrasse 25, CH-8092 Zurich, SWITZERLAND

A tool kit has been developed to analyze the empirical data of the interactive task solving behaviour described in a finite discrete state space (e.g., human-computer interaction), helping the human factors engineer to design a good interactive system. The observable sequences of decisions and actions produced by users contain much information about (1) the mental model of the user, (2) the individual problem solving strategies for a given task, and (3) the underlying decision structure. AMME (Automatic Mental Model Evaluator), the presented analysing tool kit, handles the recorded decision and action sequences and automatically provides (1) an extracted net description of the task dependent "device" model, (2) a complete state transition matrix, and (3) different quantitative measures of the decision behaviour.

1 . THE BASIC IDEA

The basic idea is to use Petri nets to model task solving behaviour. Major operations (or mappings) between Petri nets are abstraction, embedding and folding. The *folding operation* is the corner stone of our approach. A *process* (or: sequence) serves as input for this operation. An *elementary process* is the shortest meaningful part of a sequence: (s') -> [t'] -> (s'') where s' is the pre-state and s'' the post-state of the transition t'. Folding a process means mapping S-elements onto S-elements, T-elements onto T-elements while keeping the flow (F-) structure constant. This gives the structure of the performance net (see Figure 1), where each state corresponds to a system context, each transition to a system operation. The aim of the folding operation is to reduce the number of S- and T-elements of an observed task solving process to a minimum, giving the logical *task structure* (or: net). Hence, folding a process means extracting the embedded net structure while neglecting the amount of repetitions as well as the sequential order of actions.

2 . THE ARCHITECTURE OF THE ANALYSING TOOL KIT

The whole system of our analysing tool kit consists of seven different programs:
(1) An interactive *dialog system* with a logging feature, generating the task solving process description. This description should be automatically transformed to a logfile with an appropriate syntactical structure (see Table 3). However, a logfile can also be hand written by the investigator (e.g., based on protocols of observations).
(2) The *net generation program* AMME, extracting the interactive process sequence and calculating different quantitative measures of the generated net. AMME needs three input files: (i) a complete system description on an appropriate level of granularity (e.g., the state list and the pre-/post-state matrix in Table 1), (ii) the interactive process description (e.g., the logfile in Table 3), and (iii) a support file for the graphic output ("defaultp.ps" is part of the tool kit). AMME produces five different output files: (i) a *protocol file* (*.pro") with different quantitative measures of the process and of the extracted net, (ii) a *Petri net description file* ("*.net") in a readable form for the Petri net simulator PACE, (iii) a plain text file ("*.ptf") with the *connec-*

tivity matrix for KNOT, (iv) a plain text file ("*.mkv") with the *probability matrix* for the Marcov chain analysing software SEQUENZ, and (v) a PostScript file ("*.ps") to print the *net graphic* for pattern matching 'by hand'.
(3) The *Petri net simulator* PACE, being a commercial product. It is implemented in Smalltalk 80 and consists of a graphical editor and an interactive simulator with graphical animation. PACE can deal with hierarchical nets, refinement of T- and S-elements, timed Petri nets as well as stochastic Petri nets. Smalltalk 80 standard classes are available for token attributes.
(4) The *net analysing program* KNOT, computing 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.
(5) The Marcov analysing software SEQUENZ, offering a method to compare user triggered sequences. These sequences are transformed into first-order Marcov-chains. Similarity between such lattices can be directly obtained by summation of the differences between lattice-cells. Also the resulting distances provide an input to the MDS models.
(6) Any Postscript interpreter (e.g., Ghostscript) that can read and print the output file *.ps.
(7) Any text processing software supporting pure ASCII files, *.pro.

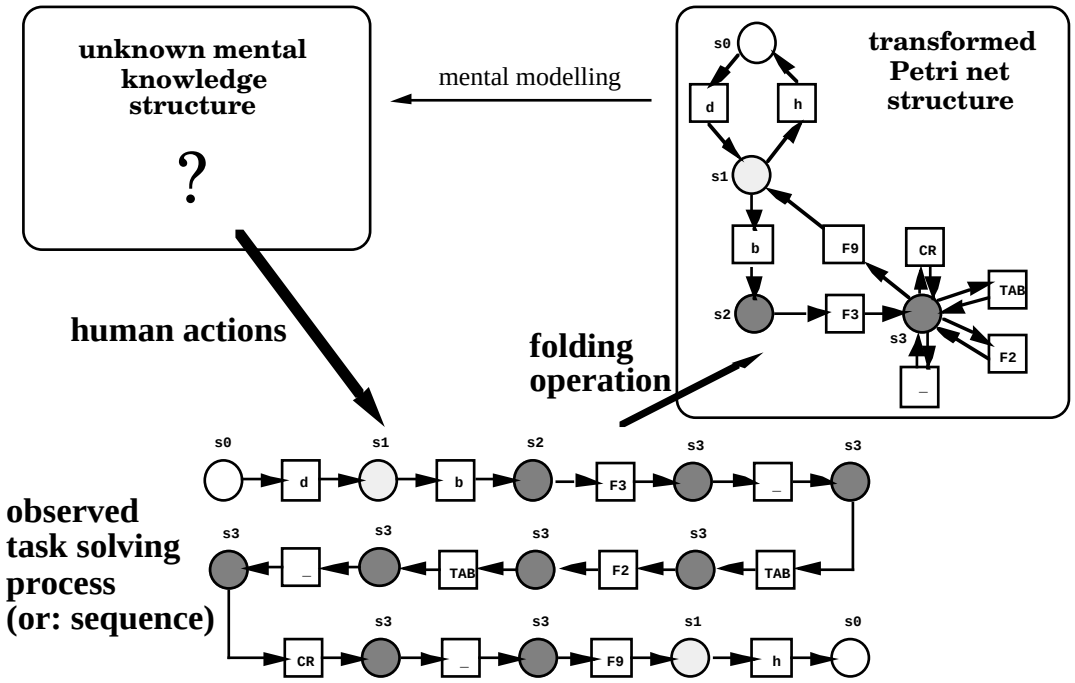

Figure 1: The 'folding' operation as the basic idea of AMME.

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. A further restriction is the constrained syntax of the logfiles that serve as input for AMME. To transform a given logfile to the appropriate form, several tools can be applied; like Coco/R, YACC, or any other tool that can convert text strings into other formats. AMME is freeware and available for IBM or compatible PCs (with MsWindows ≥3.0) via Internet: http://www.ifap.bepr.ethz.ch/~rauter/amme.html

3 . HOW TO PROCEED

First, the usability engineer has to describe the action space of the user in a quite simple syntax; the example in Table 1 shows one possible description. Table 2 describes what has happened on the screen. Table 3 shows the content of the logfile produced by the task solving process in Table 2.

Table 1: The content of a struture file (e.g., unix.str) that describes the system with all relevant states for the analysed tasks (terminal symbols of the syntax are in bold).

```
# 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 by 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 transitions
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     [CR] ; | END
# [Note: the operator ALL is very powerful to eliminate all unnecessary keystroke
#        events; before defining a transition with this operator make sure that there
#        is an other correct transition to leave the pre-state beforehand]
```

Table 2: The appearence of a concrete "login/out" procedure on the screen of a Unix server.

output and input on the screen during each dialog step	corresponding logfile content
initial state	LOG_KEYBD : <F 10>
UNIX(r) System V Release 4.0 (marshall)	LOG_MESSAGE: G_1
login: rauterberg	LOG_KEYBD : rauterbergCR
Password:	LOG_KEYBD : coraCR
Login incorrect	LOG_MESSAGE: G_2
login: rauter	LOG_KEYBD : rauterCR
Password:	LOG_KEYBD : coraCR
Last login: Wed Feb 7 19:16:57 from rota.ethz.ch	LOG_MENUE : M_3
Sun Microsystems Inc. SunOS 5.4 Generic July 1994	LOG_MESSAGE: G_4
Wed Feb 7 19:17:54 MET 1996	LOG_MESSAGE: G_5
You have no mail.	LOG_MESSAGE: G_24
marshall:/export/home/rauterberg!51> ls -l	LOG_KEYBD : ls -lCR
total 2	LOG_MENUE : M_16
drwxr-xr-x 7 rauter ifap 512 Feb 1 20:28 mac/	LOG_MENUE : M_17
marshall:/export/home/rauterberg!52> x	LOG_KEYBD : xCR
initial state	

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

The tool kit AMME can analyse the logfile shown in Table 3, generating a Petri net as shown in Figure 2. The current version of AMME is bound to a logfile syntax, appearent in Table 3. Five possible logfile events can occur: "LOG_KEYBD:","LOG_MESSAGE:", "LOG_MENUE:", "LOG_USRTIME:" and "LOG_SYSTIME:".

Table 3: The complete user logfile, showing a "login/out" task.

```
LOG_KEYBD  : <F 10>
LOG_USRTIME: 1,0
LOG_MESSAGE: G_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  : <CR>
LOG_USRTIME: 1,0
LOG_SYSTIME: 9,0
LOG_KEYBD  : c
LOG_USRTIME: 6,0
LOG_KEYBD  : o
LOG_USRTIME: 1,0
LOG_KEYBD  : r
LOG_USRTIME: 1,0
LOG_KEYBD  : a
LOG_USRTIME: 1,0
LOG_KEYBD  : <CR>
LOG_USRTIME: 1,0
LOG_SYSTIME: 49,0
LOG_MESSAGE: G_2
LOG_SYSTIME: 8,0
LOG_KEYBD  : r
LOG_USRTIME: 68,0
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  : <CR>
LOG_USRTIME: 1,0
LOG_SYSTIME: 9,0
LOG_KEYBD  : c
LOG_USRTIME: 4,0
LOG_KEYBD  : o
LOG_USRTIME: 1,0
LOG_KEYBD  : r
LOG_USRTIME: 1,0
LOG_KEYBD  : a
LOG_USRTIME: 1,0
LOG_KEYBD  : <CR>
LOG_USRTIME: 1,0
LOG_SYSTIME: 24,0
LOG_MENUE  : M_3
LOG_SYSTIME: 2,0
LOG_MESSAGE: G_4
LOG_SYSTIME: 7,0
LOG_MESSAGE: G_5
LOG_MESSAGE: 24
LOG_SYSTIME: 41,0
LOG_KEYBD  : l
LOG_USRTIME: 12,0
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  : <CR>
LOG_USRTIME: 5,0
LOG_SYSTIME: 17,0
LOG_MENUE  : M_16
LOG_MENUE  : M_17
LOG_SYSTIME: 7,0
LOG_KEYBD  : x
LOG_USRTIME: 81,0
LOG_KEYBD  : <CR>
LOG_USRTIME: 6,0
```

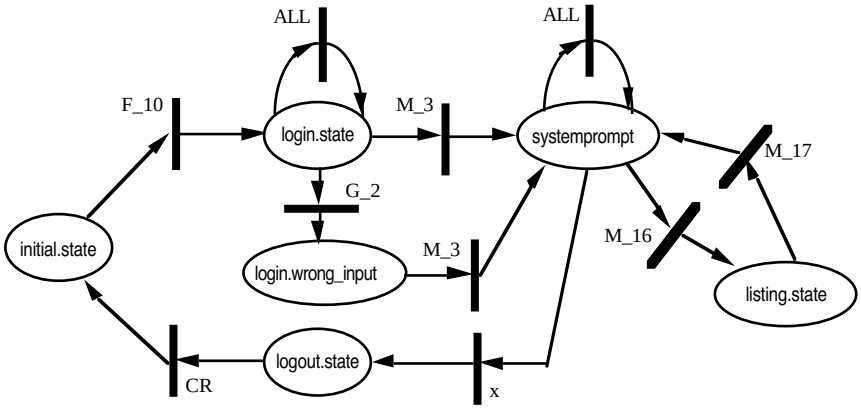

Figure 2: The generated Petri net for the given "login/out" example.

With the given tool kit AMME, we can analyse and model human behaviour in a straightforward way. Using the Petri net in Figure 2 as a model for the user task solving behaviour, we can reproduce a mean of 43% of the observed behaviour in Table 3 (see [1]).

REFERENCE

[1] Rauterberg, M. (1995). From novice to expert decision behaviour: a qualitative modelling approach with Petri nets. In Y. Anzai, K. Ogawa & H. Mori (Eds.), Symbiosis of Human and Artifact: Human and Social Aspects of Human-Computer Interaction--HCI'95 (Advances in Human Factors/Ergonomics, Vol. 20B, pp. 449-454). Amsterdam: Elsevier.

Advances in Human Factors/Ergonomics, 21B

Design of Computing Systems:

Social and Ergonomic Considerations

Proceedings of the Seventh International Conference on Human-Computer Interaction, (HCI International '97), San Francisco, California, USA

August 24-29, 1997

Volume 2

Edited by

Michael Smith
University of Wisconsin, Madison, WI 53706, USA

Gavriel Salvendy
Purdue University, West Lafayette, IN 47907, USA

Richard J. Koubek
Purdue University, West Lafayette, IN 47907, USA

ELSEVIER SCIENCE **1997**

Amsterdam – Lausanne – New York – Oxford – Shannon – Tokyo