HUMAN PROCESSOR MODELLING LANGUAGE (HPML): ESTIMATE WORKING MEMORY LOAD TROUGH INTERACTION

Jürgen Geisler, Christoph Scheben

Fraunhofer-Institut für Informations- und Datenverarbeitung - IITB,
Fraunhoferstr. 1, 76131 Karlsruhe, Germany
{juergen.geisler, christoph.scheben}@iitb.fraunhofer.de

Abstract: To operate machines over their user interface may cause high load on human's working memory. This load can decrease performance in the working task significantly if this task is a cognitive challenging one, e. g. diagnosis. With the »Human Processor Modelling Language« (HPML) the interaction activity can be modelled with a directed graph. From such models a condensed indicator value for working memory load can be estimated. Thus different user interface solutions can get compared with respect to their relative demand on working memory resources.

Keywords: human factors, human-machine interface, human perception, mental workload, memory interference, short-term memory

1. INTRODUCTION

»Reduce working memory load« recommends the last one of Shneiderman's eight golden rules for a well designed user interface (Shneiderman and Plaisant, 2005). Shneiderman argued, that »*limitation of human information processing in short-term memory requires that displays be kept simple, multiple page displays be consolidated, window-motion frequency be reduced, and sufficient training time be allotted for codes, mnemonics, and sequences of actions*«. These are more or less qualitative rules that are certainly well applicable by an experienced user interface designer. But engineers with a more shallow experience in the field of human-machine interaction who are nevertheless frequently requested to finish the technical design of their machines with a user-friendly skin often feel uncomfortable by applying such thumb rules without measurable dimensions.

Indeed the design of user interfaces is still closer to art than to science due to the lack of a clear understanding of the human brain on the one hand, and the great variety and adaptivity of human behaviour on the other. But even when there are no quantitative rules to derive a perfect user interface from a given task, and a given task split between human and machine, a quantitative comparison between various user interface solutions should be attainable. From the point of view of cognition the crucial resource is the humans working memory (short: WM). Following the well established »Model Human Processor« (MHP) from Card, *et al.* (1983) the working memory is that part of our memory where cognitive activity is located. Physical stimuli from the sense organs do activate meaningful clusters of entities in our long term memory, known as so called »chunks« that build the content of the working memory. One can say that the working memory is »the memory at work«.

Card, *et al.* (1983) derived the well known GOMS Model (Goals, Operators, Methods, Selection Rules) from the Model Human Processor. GOMS is widely used as a method to describe the effort that has to be taken from humans while conducting interaction tasks. GOMS is strong in predicting the time consumption. The time consumed from mere interaction steps is certainly an important measure to separate well suited from poorly suited user interfaces. But the losses of chunks in WM through overload seem also critical. This effect is not so well modelled, neither on the basic GOMS nor its successors like NGOMSL (Natural GOMS Language) (Kieras, 1988) or CPM-GOMS (John and Gray, 1995). One reason may be that, for less trivial tasks it is nearly impossible to determine, what information entities on the user interface are stored as *one* chunk by a certain individual. Another reason might be a different interpretation about the life cycle of chunks in the WM that is depicted in fig. 1 with a very simplified example. The WM is represented with four memory cells, each storing one chunk. Every cognitive step may load (retain) one »fresh« chunk into the WM and/or it may recall one. Following Card, *et al.* (1983, p. 393) every recalled chunk leaves its cell in WM and therefore decreases WM load (left side in fig. 1). Geisler (2006) argues that this would be comparable to a forgetting step. But forgetting can only take place by decay over time or by displacement over the capacity border. It can never be the result of a conscious step (even if e. g. NGOMSL defines a FORGET operator (Kieras, 1994)). If this is true, there must be a throughput of chunks like depicted on the right hand side of fig. 1. The chunk *C* e. g. will not vanish from WM after its recall in step `3` and therefore push chunk `A` over the capacity limit with operation `5`.

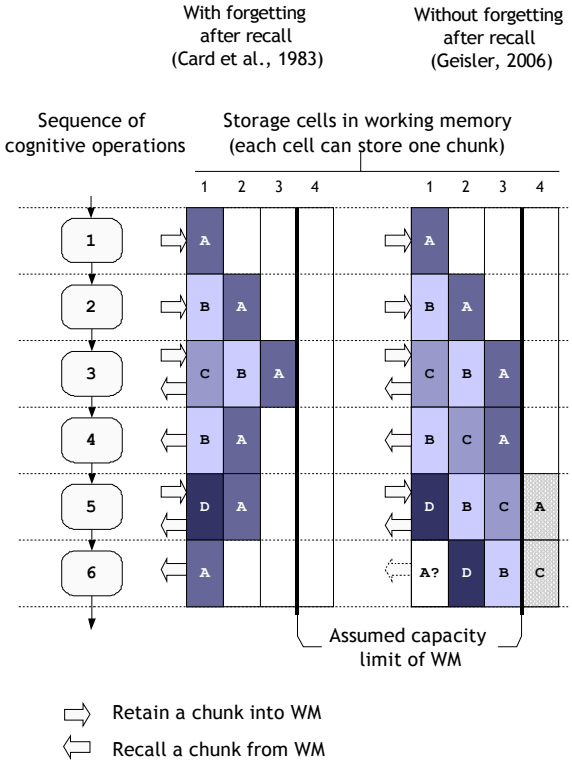


Fig. 1. Throughput of chunks in working memory at a simplified example (explanation see above).

If we assume that operation 1 in fig. 1 might be a cognitive step in a diagnosis task (e. g. to fix the source of a machine failure) and operations 2 to 5 are cognitive steps to interact with a diagnosis support tool, and operation 6 returns to the diagnosis, then in the left hand case of fig. 1 the last chunk A of the diagnosis should easily be recallable whereas it would be lost in the right hand case. So if the interpretation of Geisler (2006) comes closer to reality than that one of Card, *et al.* (1983) the throughput of chunks in the WM by interaction steps integrated into cognitive working tasks may have a higher critical effect on humans working performance. A closer look into the interaction process seems to be necessary.

2. THE HPML APPROACH

Our approach named »Human Processor Modeling Language« or short HPML is based on the three processors of the Model Human Processor from Card, *et al.* (1983): the perceptual processor that translates stimuli from the sense organs into a physical representation in the sensory image stores, the cognitive processors, that interprets them as chunks and generates new chunks in the working memory with correspondence to the long-term memory and finally the motor processor that uses certain chunks to initialize muscle activity. A HPML model is drawn as a directed graph with the processor steps as nodes and directed edges between them. Fig. 2 shows the basic scheme of HPML.

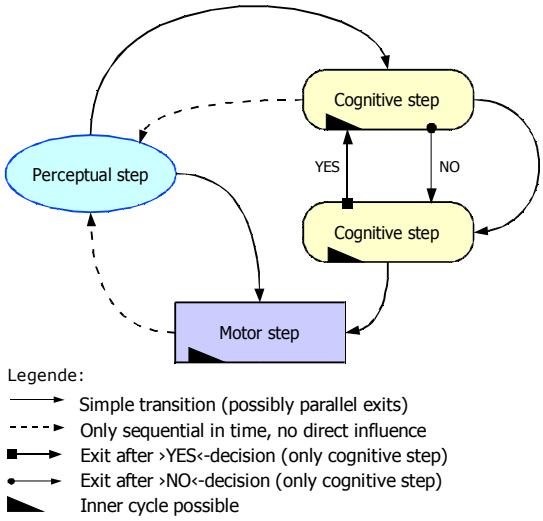


Fig. 2. Basic scheme for an HPML graph from Geisler (2006).

The steps of the cognitive processor can be tied together not only with a simple transition. If a cognitive step meets a decision, this will be indicated by special arrows dependent of the result of the decision: ›yes‹ or ›no‹. Thus one can build a kind of flowchart from the HPML elements. A special symbol is the black wedge on the bottom of cognitive and motor steps. It indicates that there are sequences of steps possible that will not be modeled in an explicit manner.

Fig. 3 gives a simple example for an HPML model. The working task shall be the visual search for a certain target, here symbolized as an airplane silhouette. Fig. 4 shows a sequence of steps from the flow chart model of fig. 3, stringed on a thread with the respective activity inside the WM. At the beginning the observer's perceptual processor writes the pattern just focused with the eyes as a chunk into the first cell of the WM. Then the cognitive processor takes this chunk and compares it with the prestored pattern of the target (aircraft silhouette). In this case the answer to the question »is this the target« is NO! And this NO is a new chunk, produced from the cognitive step (indicated here by the arrowhead from the two framed chunks into cell 1). In case of NO the next cognitive step will be to plan the following spot to fix on the screen. Therefore the also prestored chunk for the just focused position is used and results in a chunk for the change of visual fixation, indicated by the arrow in the circle. This chunk is finally used by the motor processor to execute the eye movement.

If we continue this thread with the assumption, that the target would not be found over a long period, the chunk for the target pattern, which can be seen as the representative of the search task in the WM, will not be displaced over the 6th cell. And this is still inside the working memories capacity limit of 7 ± 2 chunks (Card, *et. al.*, 1983). Additionally this chunk will be refreshed by its use from the cognitive steps and therefore won't decay over time.

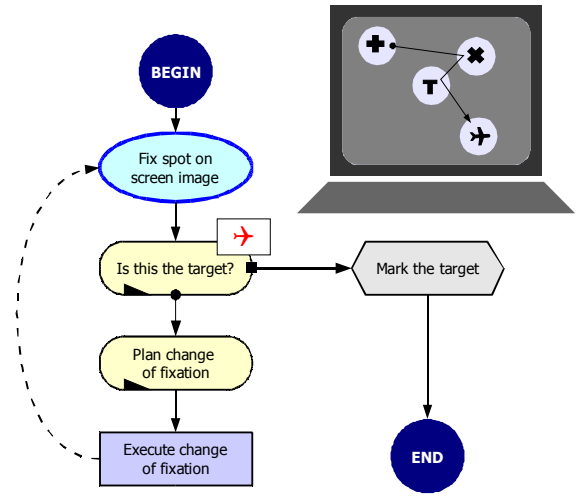


Fig. 3. Simple HPML model for a visual search.

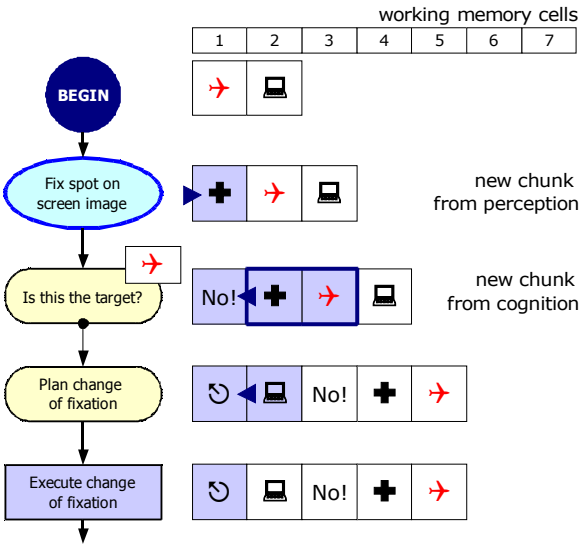


Fig. 4. Some subsequent steps on a thread and their interaction with the working memory.

The situation changes from the moment on some operations have to be carried out on the system, e. g. adjustment of brightness. This kind of task shall be called »auxiliary task«. Fig. 5 shows the case if the target cannot be found because lets say the picture on the screen is too dark. The rhomb symbol for the auxiliary task is a placeholder for the HPML model of the distinctive system operation. And this model does not only depend on the nature of the auxiliary task. It depends heavily from the interaction technique available on the human-machine interface. Fig. 6 shows the model for stepwise brightness adjustment by speech input. Coming from the working task the user first forms the necessary command mentally with a cognitive step (think: »Brighter«). As indicated by the black wedge on the bottom, this may take more than one cognitive step. As soon as the word is selected mentally, the motor processor initiates speaking.

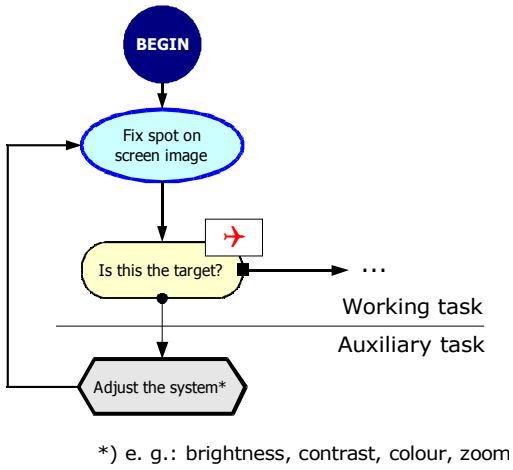


Fig. 5. Interrupt by an auxiliary task.

The now again perceived screen image (no fixation change is necessary!) is then cognitively checked whether the brightness did really change. If it did, we can move to the initializing step in the working task (here: »Is this the target?«). This is symbolized by a dashed border. If the pattern on the screen can again not be identified as the target, and the image is still to dark, the adjustment cycle starts from the beginning.

The model for speech input seems rather simple if we compare it with that one for brightness adjustment with a virtual slider on our graphical user interfaces. The respective model is depicted with fig. 7. Because it is to complex to be displayed here in full detail only the coarse structure in four blocks shall be explained. For further details see (Geisler, 2006).

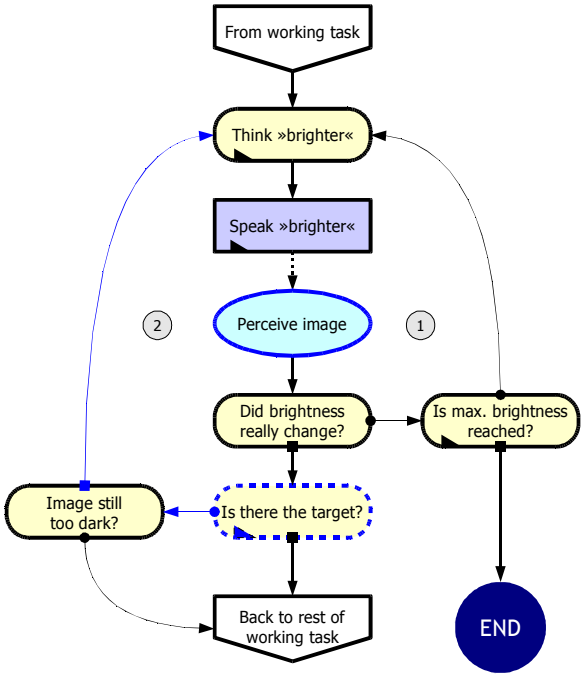


Fig. 6. HPML model for a stepwise brightness adjustment by speech input (modif. from Geisler (2006)).

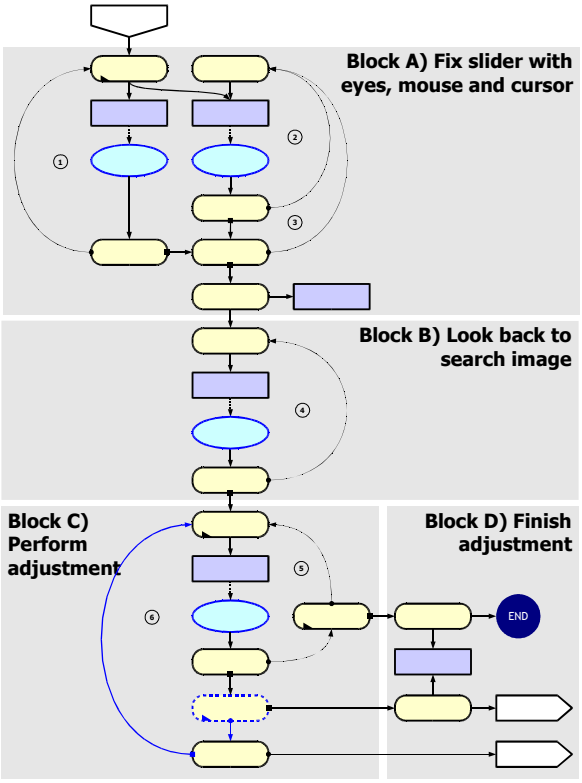


Fig. 7. HPML model for brightness adjustment with a GUI slider (modif. from Geisler (2006)).

The interaction cycle starts with block A, where the navigation of the mouse cursor to the slider grip is modeled. Watch, that the eyes have to be busy for this, and move their fixation away from the working task image. Once the slider is gripped by holding down the mouse button, the eyes have to move back to the image. This is modeled in block B. Now, with the eyes again on the possible target, the adjustment is carried out by dragging the mouse (block C), and with block D it will be finished.

The difference in complexity between speech and slider input is obvious. The more perceptual and cognitive steps we need between two recalls of the target chunk for mere system interaction, the higher is the probability that this chunk is thrown out of the working memory because of the additional chunks needed for interaction. This may cost only a few recall steps if the target pattern is persistent in long term memory. But it will be catastrophic for the working task if the pattern is volatile, e. g. because it was first acquired shortly before the adjustment becomes necessary. In this case it would get completely lost. If we presume that the complexity of the HPML model for a certain interaction has some relation to the working task performance by displacing the target chunk possibly out of the WM we ask to what degree the speech interaction is better than the one with the slider. And if we have the choice between several interaction techniques: could we bring them into an order with respect to their effect on working memory load? One simple idea is just to count the perceptual and cognitive nodes in the graph between the entrance from the working task, and the next opportunity, the target chunk is used again. But simple counting would not be sufficient because there are cycles inside the graph caused by the yes/no decisions of some cognitive steps. Through such cycles a node can be touched more than once, causing a farther displacement of the target chunk. In order to consider this effect, every perceptual and cognitive node in the model will be counted with one plus the number of cycles, that it is part of. This number shall be called the *cycle complexity* of a node. The cognitive step »think ›brighter‹ « in fig. 6 has e. g. the cycle complexity 2: It is part of the cycle ① and of its own inner cycle, indicated by the wedge. The sum of the cycle complexities of each relevant node in the graph shall be regarded as an indicator for the working memory load of the interaction sequence which is modeled with this graph (see Geisler, 2006). This indicator value shall be called B^*.

The suitability of such a model-based WM load indicator was confirmed with a simple experiment (see Geisler, 2006). Test persons got presented a target pattern for a very short moment (20 ms), and instantly after that they got displayed a complex search image wherein they had to recognize the target. This search image was visible only for 10 s, and became very quickly brighter or darker during that time. The test persons had to adjust brightness in parallel to the visual search. If the person found the target during those ten seconds or could at least remember it afterwards, the value Q_E for the quality of recall was counted with 1, otherwise with 0. Fig. 8 shows the result for five test persons with 24 trials for each. The mean Q_E is drawn as a function of the cycle complexity B^* predicted from the HMPL models for the respective interaction techniques used during the trial. Even if the values for single test persons scatter around the mean values, the linear regression shows that there is sufficient correlation to take B^* as a coarse model-based indicator for the decrease in working task performance by working memory overload through interaction.

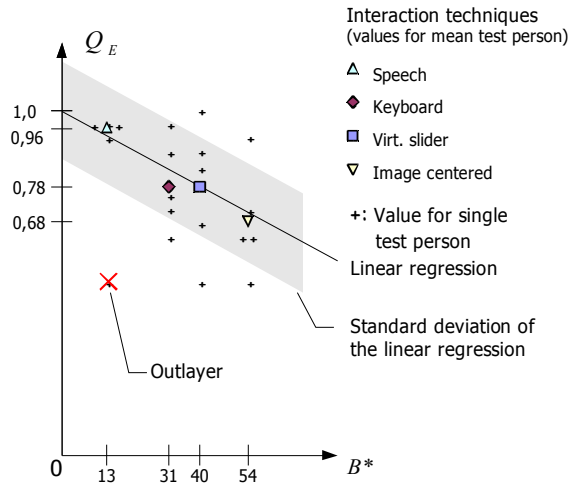


Fig. 8. Experimental result for the recall quality Q_E as function of the WM load indicator B^* from the HMPL model (modif. from Geisler (2006)).

4. TOOL-BASED MODEL COMPOSING, AND COMPUTING OF CYCLE COMPLEXITY

Modelling languages like HPML are hardly usable in practice, if one has no assisting software tool for modelling and analysis. Such a tool should

1. offer an efficient way for composing models,
2. visualise the model in an adequate manner,
3. prevent and repair syntax errors automatically as far as possible or at least report them to the user,
4. and perform calculations on the model or retrieve information from the model.

In case of the modelling language HPML the HPML-Editor (fig. 9) is a tool, which satisfies these requirements. Besides providing an easy to use user interface it checks the syntax, and is able to derive the memory load indicator automatically from the HPML model by calculating the models cycle complexity.

In the following we want to take a closer look on the formal definition of the HPM Language, and describe a way for checking HPML models. Afterwards we will focus on the calculation of the cycle complexity of a HPML model, and draw a connection between the cycle complexity and the working memory load.

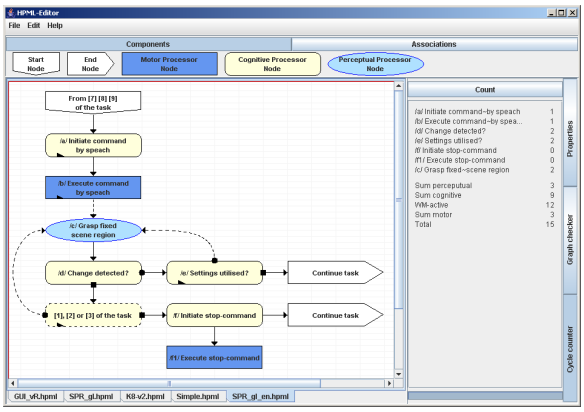


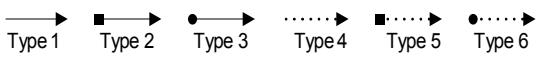
Fig. 9. User interface of the HPML Editor.

4.1 The Syntax of the HPM Language

Before we are able to check HPML models, we need to define the syntax of the HPM Language formally (in (Geisler, 2006) the definition is informal).

Definition (HPML model): *A HPML model is a simple connected directed graph G = (V, E, I) with nodes V, edges E and a mapping I : E → V x V, which subjects the following conditions:*

1. *V consists of a start node s, a finite number of MHP processors and at least one end node.*
2. *E is a finite set of connections, where a connection is an element of*



3. *I associates the edges of E with their source nodes, and their target nodes such that the requirements of fig. 10 are met.*
4. *For all v∈ V there is a path from s to v.*
5. *For all v∈ V there is a path from v to an end node.*
6. *For all cognitive processor nodes c in V the following conditions hold:*
   - *6.1. If there exists an edge e of type 2 or 5 with I(e) = (c, x) for some x, then there exists exactly one edge e' of type 3 or 6 such that I(e') = (c, y) for some y.*
   - *6.2. If there exists an edge e of type 3 or 6 with I(e) = (c, x) for some x, then there exists exactly one edge e' of type 2 or 5 such that I(e') = (c, y) for some y.*

Definition (HPM Language): *The HPM Language is defined as the set of all HPML models.*

target nodes	*source nodes*				
	start	**end**	**motor**	**cogn.**	**perc.**
start	–	–	–	–	–
end	–	–	type 1	type 1, 2, 3	type 1
motor	–	–	–	type 1, 2, 3	type 4
cogn.	type 1	–	–	type 1, 2, 3	type 1
perc.	type 1	–	type 4	type 4, 5, 6	–

Fig. 10. Allowed associations for the different types of connections.

Because the HPML Editor ensures that *I* is defined in the right way, the syntax check can be reduced to the following points:

1. Check, if there is exactly one start node *s* in the model.
2. Check, if there is a path from *s* to every (other) node.
3. Check, if there is a path from every node to an end node.
4. Check, if every cognitive processor node holds condition 6. of the definition.

These four conditions can be checked easily by an algorithm: 1. can be checked directly by logging every insertion and deletion of start nodes, 2. and 3. can be checked by depth-first search and 4. can be checked in the way the definition implies.

4.2 Computing the cycle complexity

Before drawing a connection between the working memory load, and the cycle complexity, let's define the term »cycle complexity« formally:

Definition (*nCycles*(*v*)): *nCycles(v) is defined as the number of simple cycles in G, which contain v.*

Definition (cycle complexity): *The cycle complexity is defined as*

$$\sum_{v \in V} nCycles(v)$$ [1].

Thus, the cycle complexity is an indicator for the »complexity« of a HPML model. Like mentioned above, we can derive the indicator B^* for the working memory load by summating 1 for each MHP processor in V to the cycle complexity, *and* additional 1 for every inner cycle symbol (the wedge symbol) on a MHP processor. Thus, the problem of deriving B^* can therefore be reduced to the problem of calculating $nCycles(v)$.

Unfortunately, calculating $nCycles(v)$ is NP-hard (Valiant, 1979). On the other hand, there are algorithms which are able to enumerate all simple cycles C of a Graph G in $O((|V|+|E|)\cdot(|C|+1))$, e. g. the algorithm of Tarjan (1973). (Note, that a graph may contain an exponential number of simple cycles.) The algorithm of Tarjan can be used to calculate the cycle complexity of a HPML model in the following way:

```
calculateCycleComplexity(graph G)
    C = enumerateAllSimpleCycles(G)
    n = 0
    nodeMap = ∅
    for all v∈ V
            nodeMap.put(v, 0)
    for all cycles c∈ C
            for all  v∈ c
                    n = n + 1
                    x = nodeMap.get(v) + 1
                    nodeMap.put(v, x)
    return (n, nodeMap)
```

This algorithm does not only calculate the cycle complexity of a graph $G = (V, E, I)$ in $O((|V|\cdot|C|) + (|V|+|E|)\cdot(|C|+1)) = O((|V|+|E|)\cdot(|C|+1))$, it also calculates $nCycles(v)$ for all $v \in V$. Because HPML models normally don't contain very big numbers of simple cycles (especially normally no exponential number of simple cycles), calculating the cycle complexity is no serious problem in practise.

4. CONCLUSION

With the HPML on the basis of (Geisler, 2006) a graphical language was defined to model human activity as sequence of perceptual, cognitive and motor steps according to the Model Human Processor from Card, *et al.* (1983). Under the assumption that perceptual as well as cognitive processor steps cause activity in the working memory, the working memory load with chunks should be analog to the frequency of processor activations. While this frequency cannot be predicted for a single human due to individual differences, especially in chunking, it is possible to order the models of different interaction designs for the same task with respect to their working memory load by comparing the so called cycle complexities of the model graphs. This cycle complexity can be taken as an indicator for working memory load. In (Geisler, 2006) it was shown, that this measure is suitable for the prediction of the ability to recall a chunk that has to be retained over a more or less long period of perceptual/cognitive activity.

To compute the cycle complexity of a directed graph is an NP-hard problem but can be solved analytically for a moderate number of cycles. This should cause no problem, because models of human-machine interaction with an incomputable high complexity are just for this reason not suited at all.

With the HPML-Editor one can not only design HPML models easily and syntactically correct but also directly compute their cycle complexity. So user interface designers get assistance in evaluating and improving their designs with respect to WM load model-based, without time consuming user trials. Nevertheless further theoretical advances and experiments are necessary to develop HPML to a more than only coarse approach.

[1] This definition may be misleading in terms of graph theory, because a simple cycle c is counted $length(c)$ times.

REFERENCES

Card, S., Moran, T., Newell, A. (1983). *The Psychology of Human-Computer Interaction*. Erlenbaum, Hillsdale, N. J.

Geisler, J. (2006)[2]. *Leistung des Menschen am Bildschirmarbeitsplatz: Das Kurzzeitgedächtnis als Schranke menschlicher Belastbarkeit in der Konkurrenz von Arbeitsaufgabe und Systembedienung*. Universitätsverlag Karlsruhe

John, B. E. and Gray, W. D. (1995). CPM-GOMS: An Analysis Method for Tasks With Parallel Activities. In: *Human Factors in Computing Systems (CHI 1995)*. ACM, New York, 393 - 394

Kieras, D.E. (1988). Towards a practical GOMS model methodology for user interface design. In: *M. Helander (Ed.), Handbook of human-computer interaction*. Elsevier, Amsterdam, 67-85

Kieras, D. (1994). *A Guide to GOMS Task Analysis*. University of Michigan: Technical Report, Spring 1994

Shneiderman, B., Plaisant, C. (2005). *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Fourth Edition. Addison-Wesley.

Tarjan, Robert (1973). Enumeration of the elementary circuits of a directed graph. In: *SIAM Journal on Computing Vol. 2 Issue 3*, 211-216. SIAM, USA.

Valiant, Leslie G. (1979). The Complexity of Enumeration and Reliability Problems. In: *SIAM Journal on Computing Volume 8 Issue 3*, 410-421. SIAM, USA.

[2] »Human performance at computer screens: short-term memory limits human's capacity in the competition of working task and system operation« (English edition in preparation)