Using Adaptive Layout Calculation to Handle the Visual Chaos in Graphical User Interfaces: A Retrospect on the A^{2}DL–Project

Stefan Stille[1] and Rolf Ernst[2]

[1] Steinhoff Competence Center, Langebrügger Straße 5, D–26655 Westerstede, Germany
[2] Technical University Braunschweig, Institute of Computer Engineering, Hans–Sommer–Straße 66, D–38106 Braunschweig, Germany

Abstract. The aim of our project A^{2}DL (Adaptive Automatic Display Layout[1]) was to develop concepts for the usage of layout algorithms for adaptive automatic window management in state of the art multi–window environments. With the basic objective of minimizing the user's involvement in window management activities, our emphasis has been to relieve the user from explicitly specifying any parameters for layout algorithms. Instead, an adaptive layout system is expected to learn the layout requirements directly from a user's manual corrections of the layouts presented by the system and automaticly adapt its layout calculation accordingly. In this paper, we present the basic results of the A^{2}DL–project.

1 Introduction

The extended availability and complexity of information that users have to work on with current graphical user interfaces (GUIs), often results in a situation Kahn und Charnock [12] called 'Windowitis': there are so many windows opened on the desktop that users are not able to get an overview of the information displayed in them. The visual chaos that users perceive causes significant cognitive overhead since current windowing systems mainly provide single–window layout operations which are an answer to the needs of 80's applications [13]. The results are dramatically extended task completion times and dissatisfaction of users.

In our previous work [16], [17], [18], we have proposed an approach of Automatic Display Layout (ADL), wherein it has been suggested that layout algorithms could be used to automatically organize the windows on the screen and assist the user in his current activity. The problem of ADL has been formulated as a graph layout problem by abstracting graphical objects as nodes and semantic relationships between them as edges. The layout problem was then treated as a combinatorial optimization problem [19], [20].

[1] The A^{2}DL–project was supported by German Research Foundation (DFG) under grant no. Er 168/7–1/2

In practice, however, it turned out to be a problem that the user has to specify many input parameters for the ADL layout algorithms (e.g. the optimization criteria). The user is thus, expected to know about the intricacies of the layout computations that take place in the background. This is obviously not acceptable from the software–ergonomic point of view.

In our current research [31], [32], we have propounded the concept of A^2DL (Adaptive Automatic Display Layout) where our aim was to extend the usage of layout algorithms for adaptive automatic window management in state of the art multi–window environments. With the basic objective of minimizing the user's involvement in window management activities, our emphasis, in the notion of A^2DL, has been to relieve the user from explicitly specifying any parameters for the layout algorithms. Instead, the layout system was expected to learn the layout parameters directly from a user's manual corrections of the layouts presented by the system.

2 The Framework for Adaptivity

The layout requirements of users are generally related to the current task a user performs. Our first approach to task recognition based on the working set model by Card and Henderson [9], [2]. This model states that users use distinguishable sets of windows to perform their tasks. We chose to analyze the keyboard focus to detect the current working set of windows [31]. The window getting the keyboard focus for a considerable amount of time is an indication of its usage. Provided that the working sets are detected, we classified a vector consisting of the attributes of all windows in the working set using an ART1 neural network [3]. Its output (i.e. a unique class identifier) was interpreted as a symbolic description of the task currently being performed. This approach turned out to be inadequate [33], mainly for two reasons. At first, the keyboard focus can not be solely taken as an indication of window usage because windows were used without having the focus (e.g. for read–only activities). This usage can generally not be observed by the layout system because of the limited bandwidth of the information available. Secondly, the use of neural networks for this kind of task recognition is generally prone to errors since they do not allow a detailed analysis of their behavior in a complex environment [5]. In particular, an ART1 network builds up complex dependencies between its class prototypes and the input vectors considered members of the class. The granularity of its classifications could therefore hardly be adjusted.

Consequently, we developed the AIW model (for a complete description see [33]) that describes the applications and windows needed for the execution of users' external ('real world') tasks. We presume that, in contemporary computer systems, users define their external tasks keeping the applications and windows provided by the system in mind (cf. [24]). We therefore assume that main tasks are first decomposed into 'application level tasks' (ALTs)

which allow users to determine the application(s) they have to use in order to perform a given main task. ALTs are further decomposed into 'window level tasks' (WLTs) which directly correspond to a set of windows on the screen. The windows of a WLT are, from the layout system's point of view, a semantic unit [33]. The attributes of the windows corresponding to a WLT are defined in an AIW description. At runtime, the layout system can recognize the WLTs a user *is able to perform with the currently open windows*, by comparing the attributes of these windows with those in an AIW description.

The real time task switching behavior of users is not specified in an AIW description since it is inherently unpredictable in a windowing system. However, the AIW model defines the activation states which WLTs can have during parallel task execution taking results from cognitive psychology into accout (cf. for example [1], [23]). Only one WLT at a time can be in 'active' state (i.e. consciously be performed) due to the limited capacity of short term memory [22]. All other WLTs are either 'backgrounded' (i.e. performed by the system under 'unconscious attention' from the user [23]) or 'stopped' (i.e. not performed at all). As a consequence of their limited cognitive resources, users coordinate their external tasks [10], i.e. change their activation states. It can be concluded that user oriented window operations can only be realized by supporting these task switches efficiently. Hence, in A^{2}DL, WLT objects (i.e. task objects with a mapping to the related windows) and coordination operations for WLT objects are added as GUI concepts. This enables users to directly communicate with the layout system in terms of the external tasks they want to perform. The layout system on the other side can exactly determine the currently active task and nows the context for layout calculations and the adaptation of layout requirements (figure 1).

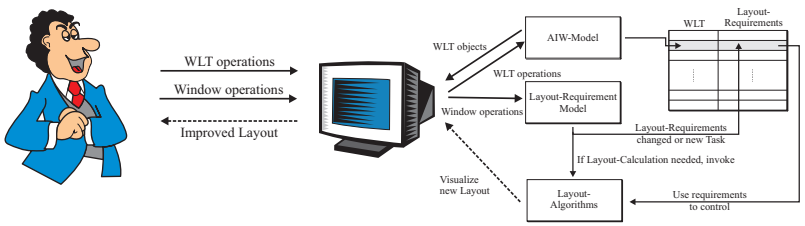

Fig. 1. Basic approach to adaptivity in A^{2}DL

3 A^{2}DL's Basic Layout Strategy

We decided to use overlapping windows rather than a tiled windows approach (cf. [4], [30]) which was also used in ADL [19] . The most important reason for our decision is that overlapping windows provide better support for parallel display of information [6], which is one of the main concerns of our research. In

addition, all contemporary commercial window systems use the overlapping layout strategy [13] so that the concepts of A^{2}DL can be applied to them.

Another basic layout principle in A^{2}DL can be derived from the user's fundamental layout requirements for windows corresponding to WLTs with specific activation states. The windows of the active WLT have to be clearly arranged on the screen since they are needed by the user for his current activities. Windows of backgrounded WLTs should be arranged as clearly as possible since the user has to check for results from time to time and then probably switch to this task. It is obvious that the windows of the active WLT have higher priority than those of backgrounded WLTs. To match these fundamental requirements, we introduce primary and secondary layout areas as shown in figure 2. Windows of stopped WLTs need not be displayed on the screen, so they can be removed (not destroyed!) by the layout system because their WLT objects still remind the user of the unfinished task[2]. For this reason, we do not use any icons for single windows in A^{2}DL.

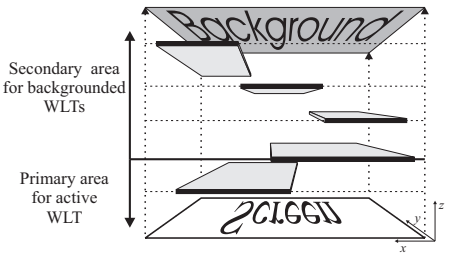

Fig. 2. Separation of z–dimension into main and secondary area. This basic layout strategy allows to place the windows of the active WLT with higher priority than those of backgrounded WLTs. The number of slots in the main area is always adjusted to the number of windows of the active WLT.

4 Identifying Layout Examples

The layout system must be able to determine the layout wishes of users by analyzing their normal windows operations (cf. figure 1). The following operations for windows are presumed to exist in a windowing system (cf. [25]): Move, Resize, Maximize/Minimize (bring to full screen size/original size), Lower/Raise (bring window to back/front of window pile). Provided that the currently active WLT is known from WLT operations, these operations are interpreted as follows:

- *Move* indicates that the current layout is not clearly arranged. The resulting layout is to be interpreted as a layout example.
- *Resize* at first indicates that the size of a window is inadequate. But it is also possible that the user thought that the layout is not clearly arranged and tried to solve this problem by changing a window's size. Therefore, the resulting layout is to be interpreted as a layout example.

[2] This is a basic requirement for window systems [23], [2], [14], [21] (see also Norman's principle of 'knowledge in the head and in the world' [27])

- *Maximize* and *Minimize* are only used to temporarily change window sizes (maybe because of some special window contents). It is therefore assumed that these operations are used to actually work on a task and do not reflect changed layout wishes.
- *Lower* and *Raise* are also used to work on a task without changing the context because the framework for efficient task switching is given by WLT operations. Therefore, these operations do not reflect changed layout wishes.

5 Cost Functions in A^{2}DL

In ADL, cost functions for layout calculation were proposed [20]. Static cost functions quantify characteristics of the current layout (e.g. overlap of windows/window titles, changes of shape/size of windows from their desired size/shape, etc.). Dynamic cost functions, on the other hand, quantify changes of the current layout relative to the last layout arranged on the screen – also called a 'reference layout' – (e.g. changes of absolute and relative positions of windows). They are important because the topological consistency of subsequent layouts is a substantial ergonomic requirement for all layout calculations [26], [31], [4].

In A^{2}DL, each cost function now exists in two versions: one that takes only windows in the primary area into account and one that takes windows from both areas (primary and secondary) into account. This diversification is necessary because of the high importance of windows in the primary area which raised the demand for an explicit control of their layout characteristics.

To analyze the distribution characteristics of cost functions, we generated stochastic layouts by randomly modifying the characteristics of given graph abstractions of screen contents. More specifically, a significant number (10000) of random, valid layouts were generated and the results of all cost functions were logged for each graph. For dynamic cost functions, we repeated these tests with a number of different reference layouts.

We found that all cost functions (static and dynamic) are approximately normal distributed over the whole solution space of the layout calculation. Given that $N(c) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(c-\mu)^2}{2\sigma^2}}$ is the density function of a cost function, we were able to show that mean μ and standard deviation σ depend on the graph structure (i.e. the number and size of nodes and the number of edges between them).

6 User Requirements and optimization goal

6.1 Abstraction of Static Layout Requirements

Given that a user provides a layout example by arranging the windows on the screen, we calculate the values $c_{d,\mathrm{stat},i}$ (called 'desired costs') of all static cost

functions $C_{\text{stat},i}$. The resulting vector $\boldsymbol{c}_{d,\text{stat}} = (c_{d,\text{stat},1}, \ldots c_{d,\text{stat},n})$ quantifies all (static) layout characteristics that are considered in A^{2}DL and is therefore a sufficient abstraction of user requirements. It is obvious that the user will not be able to perceive every deviation from the desired costs in a calculated layout. We therefore assume that a deviation of $\pm\Delta c_{\text{stat},i}$ from the desired costs is acceptable.

The probability $N(c_{d,i})$ of the desired costs $c_{d,i}$ for a specific static cost function C_i allows to estimate how hard the user must have tried to arrange a layout with these costs. If $N(c_{d,i})$ is small, only few layouts with the desired characteristics exist. Consequently, the user has focused on improving[3] this layout characteristic and will most likely only accept a small Δc_i. On the other hand, if $N(c_{d,i})$ is large, many layouts with the desired characteristics exist. The user did obviously not care about this characteristic and will therefore accept a larger Δc_i.

As a measure for the importance, the desired costs c_d have for the user, we use the following function:

$$I(c_d) \stackrel{\text{def}}{=} \frac{N(c_d)}{N(\mu)} = \frac{\frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{(c_d-\mu)^2}{2\sigma^2}}}{\frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{(\mu-\mu)^2}{2\sigma^2}}} = e^{-\frac{(c_d-\mu)^2}{2\sigma^2}}. \tag{1}$$

The maximum of I is obviously 1 for $c_d = \mu$. In addition, it is $I(c_d) < 1$ for all $c_d \neq \mu$. So I is actually a reciprocal measure for the importance.

The acceptable deviations from the desired costs for static cost functions are described by $\Delta c_{\text{stat}}(c_d)$ which is defined as:

$$\Delta c_{\text{stat}}(c_d) \stackrel{\text{def}}{=} \max(I(c_d) \cdot (k_{\text{stat}} \cdot \sigma)\,, \Delta_{\min}). \tag{2}$$

The parameter $\Delta_{\min}$ denotes the cost difference, the user can not perceive at all. It depends on the implementation of a cost function and is inherently small compared to μ and σ. The largest deviation $(k_{\text{stat}} \cdot \sigma)$ depends only on μ and σ and will be accepted when $c_d = \mu$.

From the qualitative considerations in this section, we can not completely define what deviations users will find acceptable. We therefore introduce the constant k_{stat} that allows to further customize the absolute deviation. It can only be determined in practical tests which evaluate user feedback for layout calculations with different values for k_{stat}. For reasons we give in section 8, we have not performed these tests yet.

Putting all these considerations together, the formal description of the static characteristics, a calculated layout L must have, is given by:

$$\forall i : C_{\text{stat},i}(L) = c_{d,i} \pm \max(e^{-\frac{(c_{d,i}-\mu_i)^2}{2{\sigma_i}^2}} \cdot (k_{\text{stat}} \cdot \sigma_i)\,, \Delta_{\min,i}). \tag{3}$$

[3] Due to the symmetry of $N(c_{d,i})$, low probabilities also exist for $c_d \to c_{\max}$. This means that the user tried hard to generate a 'bad' layout (for example a layout where all windows overlap completely). From our practical experience, however, we can exclude this case and will ignore it for the rest of our argumentation.

6.2 Abstraction of Dynamic Layout Requirements

Generally, a layout example does not allow conclusions about the dynamic costs that a user is willing to accept. The reason is that the user manipulates the layout by successively moving or resizing windows and therefore does not care about topological consistency. Hence, we presume that the dynamic costs should be as small as possible and define that

$$\Delta c_{\mathrm{dyn}} \stackrel{\mathrm{def}}{=} \max(c_{\min}, (\mu - (k_{\mathrm{dyn}} \cdot \sigma))) \tag{4}$$

is the upper bound of costs that are acceptable for a dynamic cost function C_{dyn}. The maximum function just simply guarantees that the upper bound can not be less than $c_{\min}$, the minimal cost value for a given layout. Like k_{stat}, the constant k_{dyn} can only be determined in practical tests. The formal description of the dynamic characteristics, a calculated layout L must have from the user's point of view, is thus given by:

$$\forall j : C_{\mathrm{dyn},j}(L) \leq \max(c_{\min,j}, (\mu_j - (k_{\mathrm{dyn}} \cdot \sigma_j))). \tag{5}$$

6.3 The optimization goal in A²DL

We define the optimization goal using equations (3) and (5):

$$C_{\mathrm{total}} = \sum_i \bar{C}_{\mathrm{stat},i} + \sum_j \hat{C}_{\mathrm{dyn},j} \quad \text{with} \tag{6a}$$

$$\bar{C}_i = \begin{cases} 0 & \text{if } \; c_{r,i} \in c_{d,i} \pm \max(e^{-\frac{(c_{d,i}-\mu_i)^2}{2{\sigma_i}^2}} \cdot (k_{\mathrm{stat}} \cdot \sigma_i), \Delta_{\min,i}), \\ |c_{r,i} - c_{d,i}| & \text{otherwise.} \end{cases} \tag{6b}$$

$$\hat{C}_j = \begin{cases} 0 & \text{if } \; c_{r,j} \leq \max(c_{\min,j}, (\mu_j - (k_{\mathrm{dyn}} \cdot \sigma_j))), \\ c_{r,j} - (\max(c_{\min,j}, (\mu_j - (k_{\mathrm{dyn}} \cdot \sigma_j)))) & \text{otherwise.} \end{cases} \tag{6c}$$

The parameters $c_{r,i}$ denote the values calculated by the cost functions for a temporary layout and are called 'raw costs'. The optimization goal only depends on the cost functions' distribution characteristics which are to be calculated for the current layout structure at runtime and on the cost vector $\boldsymbol{c}_{d,\mathrm{stat}}$ which is learned by the system. The global minimum of C_{total} is 0. It will be reached if and only if a layout has the required layout characteristics and the required degree of topological consistency. It can be concluded that the layout algorithms must be able to find the global optimum even if there are only few layouts which have the required characteristics.

7 Layout Algorithms for A²DL

As we have mentioned in section 4, manual changes of z–positions of windows are to be considered as an indication of a user's work on his task. In principle,

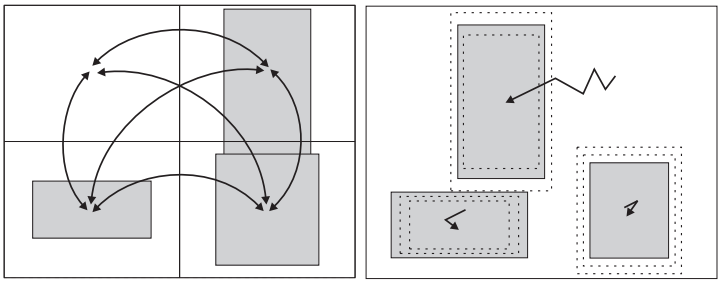

Fig. 3. In the global layout phase, a coarse grid is defined for the placement area to reduce the complexity of the solution space. In an initial step, each node is placed on exactly one grid point. Global layout changes are performed by stochastically exchanging grid positions of nodes in order to improve the overall costs of the layout. The local layout phase takes the grid based layout and calculates the final layout. Only local changes of node positions and small changes of node geometry are stochastically performed to further improve the overall costs of the layout.

layout algorithms can not support users in actually performing a task. Hence, no changes of z–positions have to be made and the 3–dimensional solution space of the overlapping layout strategy can be reduced to 2 dimensions (x and y) for layout calculation.

Layout calculations are performed in two phases with reduced solution spaces to save time. This principle was successfully introduced in ADL as a consequence of the fact that a layout calculation based on the full solution space requires calculation times of several minutes [20]. The layout algorithms are fully described in [33], a brief summary is given in figure 3.

The simulated annealing optimization strategy [15] is used for both layout phases. Control parameters define whether a temporary layout is accepted or rejected and whether the algorithm should stop because no further improvement can be expected. The annealing schedule, which specifies how the control parameters should be adapted, is therefore crucial for the quality of the algorithms. For our algorithms, we use an adaptive annealing schedule proposed in [28], which adapts the annealing process automaticly to the strucure of C_{total}.

8 Results

We performed various tests to analyze quality and behavior of our layout algorithms in practical use. For brevity, we only give the main results in this paper. For a given graph abstraction of an actual layout consisting of 6 windows with 4 semantic relationships between them, a layout L_{wish} should be calculated with static layout characteristics given by different cost vectors. The vectors were calculated from different layout examples that we manually arranged on the screen. This guarantees that at least one layout L_{wish} really

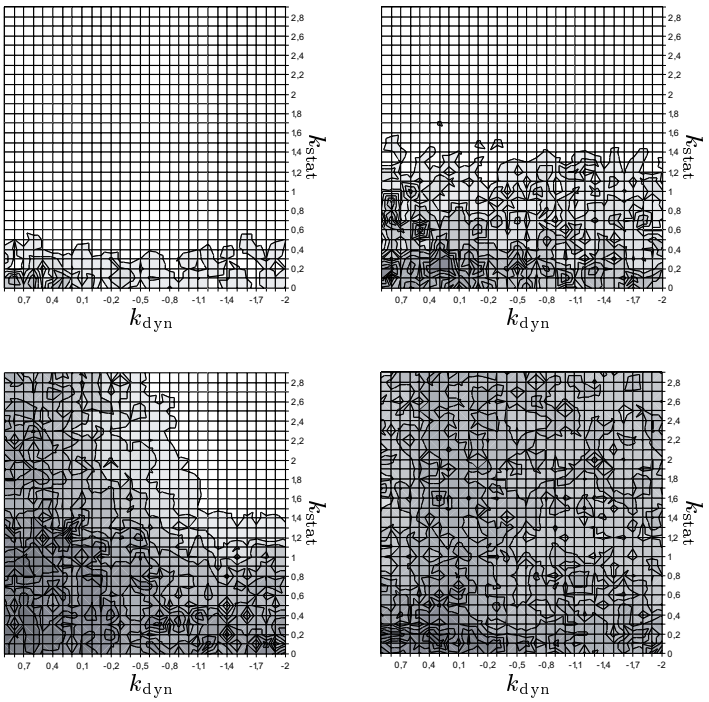

Fig. 4. Final costs for four different cost vectors.

exists. The cost vectors were selected based on the number of optimal layouts that are expected to exist, i.e. based on the complexity of the layout problem defined by them[4]. We simplified the layout problem further by taking the layout example as the reference layout for the dynamic cost functions. This guarantees that at least one layout L_{wish} exists even if total topological consistency (i.e. $\forall j : C_{\text{dyn},j}(L_{\text{wish}}) = c_{\min,j}$) is demanded.

As mentioned in sections 6.1 and 6.2, the constants k_{stat} and k_{dyn} for equation (6a) were not known at the time of the experiment. We therefore chose k_{dyn} from $[-2.0, 1.0]$ and k_{stat} from $[0, 3.0]$. For dynamic cost functions, this means that between 15,87% ($k_{\text{dyn}} = 1,0$) and 97,72% ($k_{\text{dyn}} = -2,0$) of all layouts are considered optimal during our tests. For static cost functions, the accepted deviation ranged between 0 (for $k_{\text{stat}} = 0$ and small $N(c_d)$) and σ (for $k_{\text{stat}} = 3,0$ and $c_d = \mu$, i.e. maximum $N(c_d)$). In the latter case, the cost function is practically ignored during optimization.

For each pair $(k_{\text{stat}}, k_{\text{dyn}})$ 100 layout calculations were performed. In figure 4, the average final costs C_{total} of the layout calculation are given for each parameter pair and four different cost vectors. Dark areas reflect high costs,

[4] The measure we used to determine the complexity of the layout problem is derived in [33]

white areas reflect that the layout goal $C_{\text{total}} = 0$ was reached in all 100 calculations. The complexity of the layout problem rises from the left top to the bottom right diagram.

Our experiment clearly shows that even the complexity of a relatively simple real world window layout problem is so high that it can not reliably be solved by the proposed layout algorithms. The reason is that the simulated annealing approach to optimization tends to 'get stuck' in local minima if the complexity of the layout problem gets too high. Assumed that it is possible to find an upper boundary for the complexity of real world layout problems, the calculation times for a simulated annealing algorithm with an adequate slow annealing schedule will be far from being acceptable for interactive use. Even in our simple test environment, the average calculation time for a single layout was nearly 20 seconds on an UltraSparc with a 300 MHz CPU.

9 Conclusions

The contribution of the A^{2}DL project is at first a framework which allows to realize an adaptive windowing system. The task recognition problem can now be solved by describing the relationship between external tasks and windows in an AIW model and providing WLT objects and operations for an adaptive windowing system. The semantics of manual layout operations were analyzed in the context of this model giving a detailed description of how a A^{2}DL–system should react opon their occurence. Currently, however, an AIW model must be manually generated by the user. This is relatively easy but surely not acceptable for every user. Therefore, it should be considered to build the bridge from design time models for task oriented GUI design (like TKS [11] or MAD [29]) to a runtime model like the AIW model. The designer's analytical knowledge about window usage of an application could be used to automaticly build up a runtime description for adaptive layout management.

The second major contribution of the A^{2}DL project is that the problem of adaptive window layout was formulated as a combinatorial optimization problem that allows to take user requirements into account. Two optimization algorithms based on the simulated annealing strategy were proposed to solve this problem. However, it turned out that the complexity of real world layout problems is so high that it can not be solved in reasonable time by those algorithms. Somewhat surprising, this result brings us to questions similar to those treated in the ADL project in which appropriate optimization algorithms for automatic display layout were sought. Candidates to be examined for adaptive display layout are for example genetic algorithms [8] or the tabu search paradigma [7].

References

1. John R. Anderson. *Kognitive Psychologie.* Spektrum Lehrbuch. Spektrum Akademischer Verlag, Heidelberg, Berlin, Oxford, 1996.
2. S. K. Card, M. Pavel, and J. E. Farrell. Window-based computer dialogues. In *Proceedings of IFIP INTERACT'84: Human-Computer Interaction*, Dialogue Interaction / Analysing Interactive Dialogues, pages 239–243, 1984. Reprinted in Baecker & Buxton, 1987, p. 456.
3. G. A. Carpenter and S. Grossberg. The art of adaptive pattern recognition by a self–organizing neural network. *IEEE Computer*, pages 47–88, March 1988.
4. E.S. Cohen, E.T. Smith, and L.A. Iverson. Constraint–based tiled windows. *IEEE Computer Graphics and Applications*, 6(5):35–45, May 1986.
5. G. Dorffner. *Konnektionismus: Von Neuronalen Netzen zu einer 'natürlichen' KI.* Teubner, Stuttgart, 1991.
6. Douglas J. Funke, Jeannette G. Neal, and Rajendra D. Paul. An approach to intelligent automated window management. *International Journal of Man–Machine Studies*, 38:949–983, 1993.
7. F. Glover. Tabu Search – Part1. *ORSA Journal on Computing*, 1(3):42–51, 1989.
8. Goldberg, D. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison–Wesley, 1989.
9. D. Austin Henderson and Stuart K. Card. Rooms: The Use of Multiple Virtual Workspaces to Reduce Space Contention in a Window–Based Graphical User Interface. *ACM Transactions on Graphics*, 5(3):211–243, July 1986.
10. Michael Herczeg. *Einführung in die Software–Ergonomie.* Addison–Wesley, Deutschland, 1994.
11. H. Johnson and P. Johnson. Task knowledge structures: Psychological basis and integration into system design. *Acta Psychologica*, 78:3–26, 1991.
12. M. J. Kahn and E. Charnock. How to prevent "windowitis" in your graphical interface. In *Proceedings Silicon Valley Ergonomics Conference & Exposition, ErCon '95*, pages 18–25, 1995.
13. E. Kandogan and B. Shneiderman. Elastic windows: Improved spatial layout and rapid multiple window operations. Technical Report CS–TR–3522, Human–Computer Interaction Laboratory, University of Maryland, College Park, MD 20742-3255, USA, September 1995.
14. E. Kandogan and B. Shneiderman. Elastic windows: Evaluation of multi-window operations. Technical Report CS–TR–3695, Human–Computer Interaction Laboratory, University of Maryland, College Park, MD 20742-3255, USA, October 1996.
15. S. Kirkpatrik, C. D. Gelatt jr., and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(13):671–680, 1981.
16. P. Lüders and R. Ernst. The dynamic screen – beyond the limits of traditional graphical user interfaces. In *Proceedings of 13th IFIP World Computer Congress, Hamburg, Germany*, pages 109–114, Amsterdam, The Netherlands, September 1994. Elsevier Science Publishers.
17. P. Lüders and R. Ernst. Improvement of the user interface of multimedia applications by automatic display layout. In *Proceedings of Multimedia and Networking Conference, San Jose, CA.* Multimedia and Networking Conference, San Jose, CA, February 1995.

18. P. Lüders and R. Ernst. Improving browsing in information by the automatic display layout. In *Proceedings of the IEEE Symposium on Information Visualization, Atlanta, Georgia, USA*, pages 26–33, Los Alamitos, California, USA, October 1995. IEEE Computer Society Press.
19. P. Lüders, R. Ernst, and S. Stille. An approach to automatic display layout using combinatorial optimization algorithms. *Software – Practice & Experience*, 25(11):1183–1202, November 1995.
20. Peter Lüders. *Ein Beitrag zur Verbesserung der grafischen Benutzungsschnittstelle durch das automatisierte Bildschirmlayout.* Fortschritt–Berichte VDI. VDI Verlag, Düsseldorf, 1995.
21. Thomas W. Malone. How Do People Organize Their Desks? Implications for the Design of Office Information Systems. *ACM Transactions on Office Information Systems*, 1(1):99–112, January 1983.
22. G. A. Miller. The magical Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information. *Psychological Review*, 2(63):81–97, 1956.
23. Y. Miyata and Norman D. A. Psychological issues in support of multiple activities. In D. A. Norman and S. W. Draper, editors, *User Centered System Design*, pages 265–284. Lawrence Erlbaum Associates, Publishers, Hillsdale, NJ, 1986.
24. Thomas P. Moran. Getting into a system: External-internal task mapping analysis. In *Proceedings of ACM CHI'83 Conference on Human Factors in Computing Systems*, Interface Design 2 – The Design Process, pages 45–49, 1983.
25. Brad A. Myers. A taxonomy of window manager user interfaces. *IEEE Computer Graphics and Applications*, 8(5):65–84, September 1988.
26. F. Newberry-Paulish and W. F. Tichy. EDGE: An Extensible Graph Editor. *Software – Practice & Experience*, 20(6):63–88, July 1990.
27. D. Norman. The design of everyday things, 1990. Doubleday: New York, NY.
28. R.H.J.M. Otten and L.P.P.P. van Ginneken. *The Annealing Algorithm.* The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Press, Boston, Dordrecht, London, 1989.
29. Suzanne Sebillotte. Methodology guide to task analysis with the goal of extracting relevant characteristics for human–computer interfaces. *International Journal of Human–Computer Interaction*, 7(4):341–363, 1995.
30. Ben Shneiderman. *Designing the user interface: strategies for effective human–computer interaction.* Addison–Wesley, 3. edition, 1998.
31. S. Stille, S. Minocha, and R. Ernst. A^2DL–An Adaptive Automatic Display Layout System. In *Proceedings of the IEEE 3rd Annual Symposium on Human Interaction with Complex Systems (HICS 96)*, Dayton, OHIO, USA, August 1996. IEEE Computer Society Press.
32. S. Stille, S. Minocha, and R. Ernst. An adaptive window management system. In *Proceedings of the 6th IFIP Conference on Human-Computer Interaction*, Sydney, Australia, Juli 1997.
33. Stefan Stille. *Adaptive Layoutberechnung zur anwendergerechten Layoutgestaltung in Grafischen Benutzungsoberflächen.* PhD thesis, Technische Universität Braunschweig, 1999.