

# Automatic Display Layout in Window Oriented User Interfaces

P. Lüders and R. Ernst

Institut für Datenverarbeitungsanlagen, Technische Universität Braunschweig,  
Hans-Sommer-Str. 66, 3300 Braunschweig, Germany

## Abstract

Using today's or future complex software systems like hypertext, CA-systems or electronic publishing, the user is confronted with a large amount of correlated information which is generally displayed in several windows of a window oriented user interface. He is busy selecting momentarily relevant information, arranging it on the display and, finally, he has to comprehend the displayed information with its explicit or implicit relations. As key to a systematic improvement of the user interface, we identified automatic display layout. In our experimental tool *Screen-Broker* we use combinatorial optimization algorithms like simulated annealing, genetic algorithm and force directed algorithm directed by an objective function to compute the display layout. In this approach we try to combine the experience of recent research in VLSI- and in graph-layout. While the implemented layout algorithms are general in nature, the objective function is application specific. The given results show that automatic display layout in an interactive environment is feasible.

Keyword Codes: H.5.2; D.2.2; I.3.3

Keywords: User Interfaces; Tools and Techniques; Picture / Image Generation

## 1. Introduction

Graphical representations of graph structured information are used for development and documentation in a wide range of applications. They span the gamut from visualizations of genealogical trees over the documentation of road maps to the representation of technical systems. Therefore it is an essential task for a computing system to assist the user who works with graph structured information.

Cooperative processing, multimedia or electronic publishing involve complex interactions possibly between groups of workstations and users. State of the art in software systems are programs which often use several windows in a window oriented user interface for display or user input. The user faces a



bulk of information and has to learn about the structure and contents of the actual display and, at the same time, has to find orientation in the network of correlated informations. To solve a specific problem, the user will quit or iconize less interesting windows and change the size and the position of windows containing problem relevant information. Therefore, he is busy with tedious and time consuming work and is distracted from solving the real problem.

For these reasons we regard the introduction of display layout, which shall stand for automatic sizing, placing and interconnection of graphic objects, as an essential improvement of the user interface, which will result in an increased user productivity.

In this paper we present our tool, *ScreenBroker*, as a layout tool for computing display layout as well as for computing general graph layout within a single window. It is an experimental tool to gather experience with the requirements to automatic display layout in interactive user interfaces. In this tool, the user edits a textual description of a graph in a usual text editor and a graphical representation in a second window is automatically computed. In order to reduce the amount of displayed information — especially in editing large graphs — the user is able to select momentarily interesting information by application specific filter macros. Subsequent to the execution of a filter macro, a new display layout is computed for the selected data only.

The main problem for interactive display layout is computation time. Our solution is a three phase approach, based on general combinatorial optimization algorithms like simulated annealing, genetic algorithm or force directed algorithm. In the first and the second phase a coarse raster based layout with normalized object sizes is computed. During the third phase, the adaption of arbitrary object sizes is introduced to the layout. Since this is done iteratively with incremental changes to the layout topology, the process of development can be displayed on the screen to give the user the impression of a *natural flow of objects*. Therefore, he is able to learn the display layout while it is computed.

The essential benefit of this approach is, that it is generally applicable and that it is possible to control the computational result (i.e. the layout) by the objective function. It is used within the optimization algorithms for evaluating a layout and each algorithm tries to minimize its value during the computation. Since the algorithms optimize the objective function, modifying the objective function leads to different layout results.

## 2. The Problem of Display Layout

The problem of automatic display layout is to place the elements of a hierarchical graph, given by a set of nodes (windows) and a set of edges (relations between pairs of windows) — directed or undirected, visible or invisible — on a given placement area (display). In our approach, hierarchical structure is considered with the

introduction of parent-child relations and the hierarchy is graphically represented by placing the children into the parent node.

For display layout, several requirements for the computation of layouts and characteristics of the graph structure can be outlined.

- The total number of graphical objects to be placed may be limited to around 50 in order to get an intelligible display layout.
- The size of each node (window) can range between iconisation and full screen mode.
- Graphical representations differ in layout style and layout requirements. Examples are flow charts, hierarchical schematics or window placement in hypertexts. Even for the same application, users may have varying preferences.
- To ensure continuity during a user's work, small changes in the display structure — like to open or to close a single window — have to lead to a new display with a similar layout topology. It is unacceptable for a user to have to work with a completely new layout subsequent to each layout computation. We have introduced the term *topological consistency* for this requirement.
- At last, the response time to a user request has to be in the range of a second in order to permit efficient work. In our experiment, we accept a limit of 10 seconds, because we assume a factor of 10 in computation speed by the time the approach is used in practice.

An essential task for computing a display or graph layout is the development and implementation of a function to evaluate the quality of a computed layout. Such a function is called *objective function*. Our approach to the objective function is a weighted sum of partial objectives. As a first step, we have identified several rather obvious characteristics of a good layout and implemented them in functions thereby trying to minimize the computation time. The weights of the partial objects are parameters to allow experiments and adaption to individual requirements. In a later state of the project these assumptions have to be validated in test series.

Considering the problem of topological consistency, we have divided the objective function into two major parts that are the static and the dynamic objective. While the elements of the static part evaluate characteristics of a single layout, the elements of the dynamic parts rate the characteristics of succeeding displays. We have implemented five elements to evaluate characteristics of the static part of the objective function.

- *Edge direction*: some representations, such as circuit schematics, are sensitive to edge direction. This is measured by edge length in horizontal ( $\Delta X$ ) and vertical ( $\Delta Y$ ) direction.
- *Edge Length*: as in most approaches for computing graph displays, we define the minimization of the total edge length as an optimization criterion.

- *Edge crossing*: again, like in most approaches, the number of edge crossings should be minimized.
- *Homogeneous distribution*: agglomerations of graphical objects in several parts of the placement area seem to be undesirable, therefore a homogeneous distribution of objects on the display area should be an attribute of the layout, too.
- *Visibility*: in the current implementation, overlapping of objects is not allowed to ensure the visibility of edges connecting the objects and the contents of each object.
- Additional objective function components have to be introduced to evaluate the size of objects compared to the required size, desired size and relative size.

To quantify the changes of the display in succeeding layouts, we have implemented two elements as the dynamic part of the objective function:

- *Absolute changes*: the sum of the average distances of all objects and sizes between their new and old locations is a first element to quantify the topological consistency. This results in nearby locations of the objects in subsequent figures thus preventing rapid moves on the screen. It can also be used to fix objects on the screen.
- *Relative position*: the changes in the relative positions between the objects are evaluated, too. This attribute can be evaluated by the average difference in the directions of imaginary lines connecting the centres of all nodes in subsequent displays. A high weighting of relative positions suppresses rapid changes of the topological ordering of screen objects.

### 3. Previous and Related Work

A general bibliography for graph layout algorithms is given in /1/. Examples for browsers are EDGE /2/, GRAB /3/, ISI Grapher /4/ and GERM /5/. Their placement algorithms are based on an algorithm first described in /6/. This algorithm is not suitable for general cyclic graphs with various node sizes like graphs for display layout. Furthermore hierarchy is often included, but in the case of nodes with different sizes the display is not efficiently used. The problem of topological consistency is introduced in EDGE and GRAB using the term 'similarity'. While in GRAB the topological consistency is not further treated, in EDGE it is regarded by layout constraints, i.e. relations between nodes like "A right of C" or "A above B". Such constraints, however, either totally fix the topological arrangement of nodes - in case of complete ordering - or give a partial ordering which leaves it to the layout algorithm how to place nodes without defined placement relations between them disregarding topological consistency. Therefore, such an approach is either too inflexible or it can not guarantee topological consistency. It seems to be more useful to explicitly define an objective function to directly control the algorithms with regard to topological consistency.

In some hypertext /7/ systems like CONCORDE /8/, automatic display layout is implemented to visualize the structure of the hypertext. In /8/, a simple hillclimbing algorithm is used to place equal sized nodes. Again, the problem of topological consistency is not taken into consideration. But, like browsers, hypertext systems improve the user interaction by possibilities to navigate through the graph and by filter macros to select specific information.

A general introduction to VLSI-layout is given in /9/. The interested reader, in particular in layout algorithms, is referred to /10/, /11/, /12/, /13/. The main objective of placement algorithms used in VLSI-layout is area minimization. As an approximation, the objective function is minimum wire length. This is a very simple function compared to our problem.

The general structure of the *ScreenBroker* and the examination of the implemented algorithms are described in /14/. The implementation of layout algorithms is based on work in /12/, /13/ (simulated annealing), /10/, /11/ (genetic algorithm) and /15/ (force directed placement). Some components of the objective function to direct the optimization algorithms are derived from constraints for graph display given in /16/.

## 4. Computing Display Layout in Three Phases

### 4.1. Our Approach to Display Layout

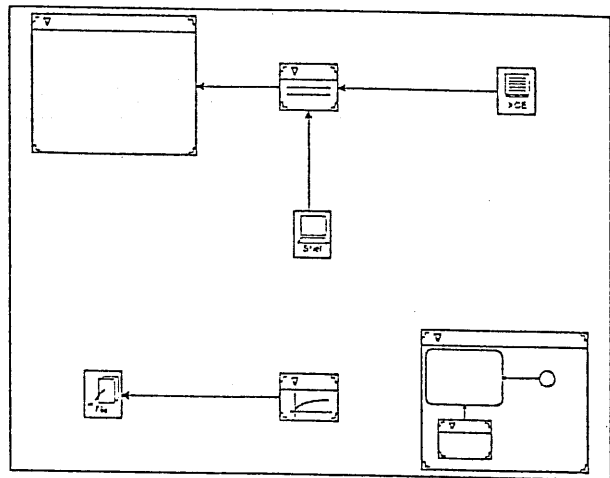
The requirement of flexibility has led us to general combinatorial optimization algorithms like simulated annealing or genetic algorithm, the restriction of computation time has forced the splitting of the computation of a display layout in three parts that are global layout, local layout and postprocessing.

To describe the three phases more precisely, let us assume the computation of a display layout for quite a simple example shown in figure 1. Graphic attributes to each node like color, form or desired size of each window are assigned either by the tools which control the windows, in a preprocessing phase of the *ScreenBroker*, or by the user itself. Let us furthermore assume that an objective function for evaluating a layout exists.

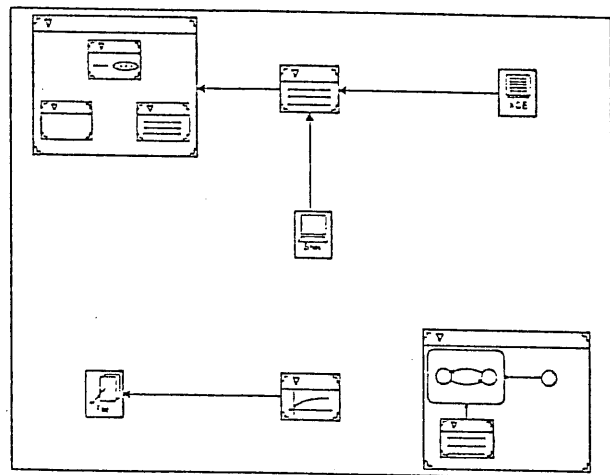
The computation of a new display layout starts with the global layout. The task of this phase is the computation of a coarse raster based layout. In order to reduce the computation time needed until the first layout is displayed, the number of nodes to be placed is decreased by disregarding nodes of limited local subgraphs (see section 4.3.1). At the end of the global layout phase the rough display layout is displayed by giving the user a first impression of the layout (see first layout in figure 1).

In the second phase, the local layout, the limited local subgraphs are placed using a look-up table approach and the graph displayed on the screen is completed. The actual result can be characterized by orthogonality and inefficient use of the screen size (see second display in figure 1).

First display of the (incomplete) layout subsequent to the global layout phase.



First display of the complete layout after the local layout phase.



Final layout computed in the postprocessing phase.

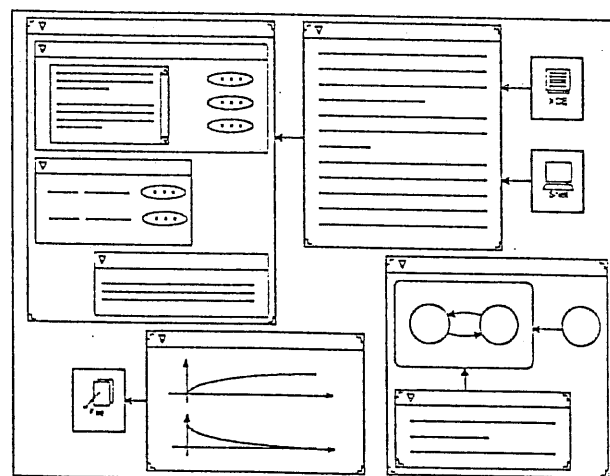


Figure 1: Computing a display layout in three phases, example.

During the postprocessing phase the displayed layout is improved by incremental changes to the layout structure. The grid locations are disregarded and the size of the nodes is increased or decreased considering the desired size assigned before. These incremental changes are displayed on the screen resulting in an impression of a *natural flow of objects* for the user.

The last phase has inspired the system's name, *ScreenBroker*. Like a broker in the stock exchange, each window has to announce a place on the screen with a desired size and the broker tries to satisfy the demands and to distribute the available placement area.

## 4.2. Objective Function

The optimization algorithms improve a layout relying on the evaluation of the objective function. Therefore, the evaluation of layout characteristics in the objective function is decisive for computing an intelligibly arranged layout. Since the objective function evaluates several characteristics of a layout, there are problems for weighting the various components, for the number of the characteristics evaluated and for the computation time for evaluating a layout.

The complexity of the objective function is essential for the computation time. It is called several thousand times during the computation process for a graph with approximately 10 — 20 nodes. Therefore, it is necessary to be very careful in defining the objective function. In the current implementation, we mainly used the static attributes of edge length and DeltaX for computing a layout. These attributes can easily be calculated, and with changing one element of the graph, only the value of the affected elements has to be corrected during the calculation of the objective function. The attribute of homogeneous distribution is ensured by the adjusted raster size. Furthermore, we have made the experience that by minimizing the total edge length the number of edge crossings tends to decrease as well. Therefore the calculation of the edge length seems to be a suitable first approach. But, like the examination of the dynamic attributes, such speculations have to be validated in test series.

## 4.3. Implementation of Layout Algorithms

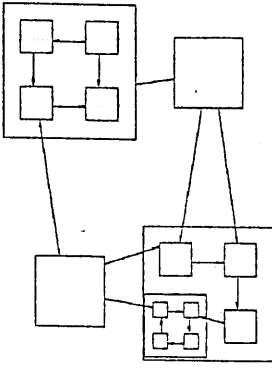
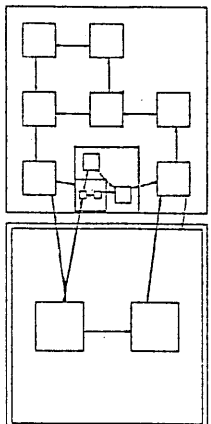
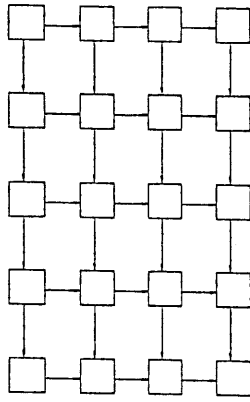
### 4.3.1. Global Layout Phase: Simulated Annealing and Genetic Algorithm

The task of the global layout phase is the computation of a raster based layout which is arranged according to the given objective function. In order to reduce the computation time until the first layout can be displayed, it is reasonable to look for possibilities to reduce the number of nodes (objects) to be considered. Therefore, we introduce the term *limited local subgraphs*. We call the set of children of a parent a limited local subgraph if the number of children is less than a number  $n$ , if there are no connections between them and nodes outside this set and if there is no further hierarchical structure within the children. We currently set  $n = 7$ . These limited local subgraphs are marked at the beginning of the global layout



Table 1.

Layouts computed by simulated annealing and genetic algorithm (Sun Sparc1+, 64 MByte main memory).

			
Sim. Annealing			
Comp. time	0.4 s	0.9 s	5.2s
No. of Iterations	633	1120	8856
Genetic			
Comp. Time	0.6 s	2.8 s	6.4 s
No. of Iterations	400	800	2800

phase and are placed subsequently during the local layout with a look-up table based placement algorithm.

In a first step, a hierarchical raster with a size adjusted to the number of nodes and to the x-/y-dimension of the screen size has to be defined. There are two essential advantages using an adjusted raster for possible locations of nodes. First, the objective function can be simplified because homogeneous distribution is ensured and an overlapping of nodes can not occur. Secondly, temperature and annealing function of a simulated annealing optimization depend on the possibilities to move a node. The reduction of these possibilities results in much faster annealing at lower temperature resulting in less computation time. A similar statement holds for genetic algorithms.

The implementation of the simulated annealing is based on a description in [13]. It melts a random system - in our case a random layout - at a 'high' temperature and then lowers the temperature in small steps. In each iteration step it tries to improve the layout depending on the given objective function with provisions to escape local minima. The improvements are achieved by movements of nodes. The temperature determines how far an element may move. The algorithm terminates by reaching a precalculated estimated optimum of the objective function or by reaching a given limit of computation time.

In contrast to the simulated annealing, which tries to optimize a single layout sequentially, genetic algorithms simulate the biological evolution using a genera-

tion of layouts. With a genetic algorithm, operators of the biological evolution like mutation, crossing of a pair of individuals and selection are simulated.

At the current state of the project, both algorithms are still used because both have their specific benefits. While using the currently raw and simple objective function, simulated annealing is faster than the genetic algorithm, because the objective function can be computed incrementally if only two nodes are changed or one node is moved. On the other hand, we found that a genetic algorithm generates layouts as good as simulated annealing thereby requiring fewer calls of the objective function. But, incremental objective function computation is inefficient in a genetic algorithm, because the crossing operator (dominant in genetic algorithms) changes a group of several nodes in one step. Considering the increasing number of workstations which allow parallel computing, the genetic algorithm is promising because of its evident parallelism. For these reasons we continue the examination of both algorithms and postpone a decision to a later stage. In table 1, examples of layouts are presented that were computed with simulated annealing and genetic algorithm. It is interesting to note that the rightmost graph, almost very regular, needs by far the most time because of the large number of possible permutations and local minima.

#### **4.3.2. Local Layout: Look-up Table Approach**

For computing a local layout using a look-up table based algorithm, two problems have to be regarded: the definition of table contents and the look-up procedure. In our implementation, a limited local subgraph is first classified by a sorted list of the number of edges summed up for each node. In the second step, an entry for this notation is searched in the look-up table. The relative topology of the nodes in the entry is adjusted to the real place and size. In addition, the given objective function and the edges between nodes of the limited local subgraph and other nodes have to be considered, too. Therefore, the entries in the look-up table are selected one by one and the value of the objective function is determined for various orientations of the placement described in the entry, and the best position found is taken as the final placement. The computation time for the local layout is neglectable in the range of milliseconds. The look-up table contents depend on the application.

#### **4.3.3. Postprocessing: Force Directed Placement Algorithm**

The display layout computed in the global and local layout phases can be characterized by its orthogonality and its little efficient use of the placement area, especially for displays of hierarchical graphs and standard object sizes. The main task of the postprocessing algorithm is the adaptation of size and placement without raster limitation.

In our approach, we use a force directed algorithm. In this algorithm, the edges of the graph are mapped to springs which are inserted between the nodes then applying Hooke's law. The force between two nodes connected by (an) edge(s)

results in the spring constant (e.g. number of edges) times the distance between the nodes. The nodes are moved iteratively in order to reduce the forces between the nodes. An implementation of this algorithm for graph layout is described in /15/. Here, attractive and repulsive forces (to prevent a contraction into one point) are used to modify a graph layout.

In order to adapt this algorithm to the hierarchical display layout, several new forces had to be introduced. The repulsive and the attractive force listed are the traditional ones for graph layout. The *border force* has to be defined with regard to the hierarchical structure of the graph and object sizes.

- *Repulsive force*: between each pair of nodes a force is defined to push the nodes apart for a homogeneous distribution. Its strength is chosen as the inverse of the distance between the nodes, therefore no overlapping of nodes can occur.
- *Attractive force*: nodes connected by (an) edge(s) are pulled together by another force.
- *Border force*: forces depending on the distance between each node and the border of the placement area in +x, -x, +y, -y direction are included. Using these forces only, nodes will be placed in the center of their placement area. Like the repulsive force, its strength is inverse to the distance to prevent a node from moving out of its placement area.

To adapt the sizes of graphical objects, which usually results in enlarging and / or reducing some nodes or in reducing some nodes to permit the enlargement of others, additional functionality has to be introduced. In order to solve this problem, the terms *inside pressure* and *outside pressure* are defined. The basic idea is that a node with an inside pressure larger than the outside pressure enlarges its size until both pressures have the same value. The values are defined as the following:

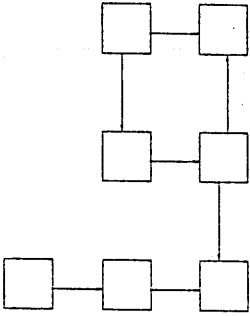
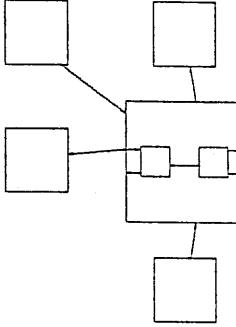
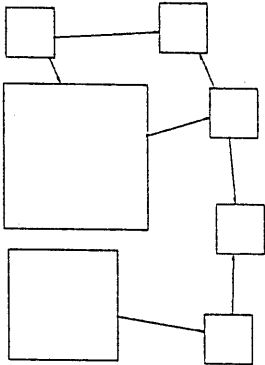
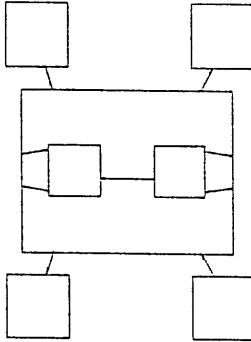
- *Inside pressure*: sum of all border forces of its children or a value depending on the desired size given by visualization rules /14/.
- *Outside pressure*: sum of all border and repulsive forces effected on the actual object.

Because of the complexity of force calculation, we have split the total graph into several parts, which are improved independently. Each part consists of the set of nodes with the same parent. Therefore, the graph parts have no hierarchical structure. Note, that although the parts are improved independently, they are connected by the inside and outside pressure.

The algorithm starts with the calculation of forces and pressures effective on each object during the initialization phase. Within the main iteration loop, all systems are first improved by the move of a node and, then, by altering the size of a node. The move is determined by searching for the object with the largest

Table 2.

Examples of computed layouts before and subsequent the postprocessing phase (Sun Sparc1+, 64 MByte main memory).

Computed layout after global layout phase		
Computed layout after postprocessing		
Time (algorithm)	1.4 s	1.8 s
Time (display)	1.5 s	2.7 s
No. of iterations	53	80

effective force. An incremental move in order to lessen the force is executed on this object. The step size of the move is given by a user defined parameter adjusted to the size of the actual system. In a similar mode of operation, an object is selected for size modification. Here, the result of the subtraction of outside pressure and inside pressure is calculated for all nodes. The node with the maximum absolute difference is resized, i.e. depending on the sign enlarged or reduced. Again, the step size is given by the user. It is to be mentioned that at the current state of work, enlargement of object sizes by the same factor for both directions is the only possibility of altering object sizes. The main iteration loop stops if a given number of iterations is reached or the improvement between subsequent iterations falls under a given limit. In table 2 two simple examples are shown before and after the computation of the final layout.

At last, forces are introduced to regard the topological consistency, i.e. an attractive force between the actual location of a node and the location in previous displays and a force to keep the relative positions between pairs of nodes. But

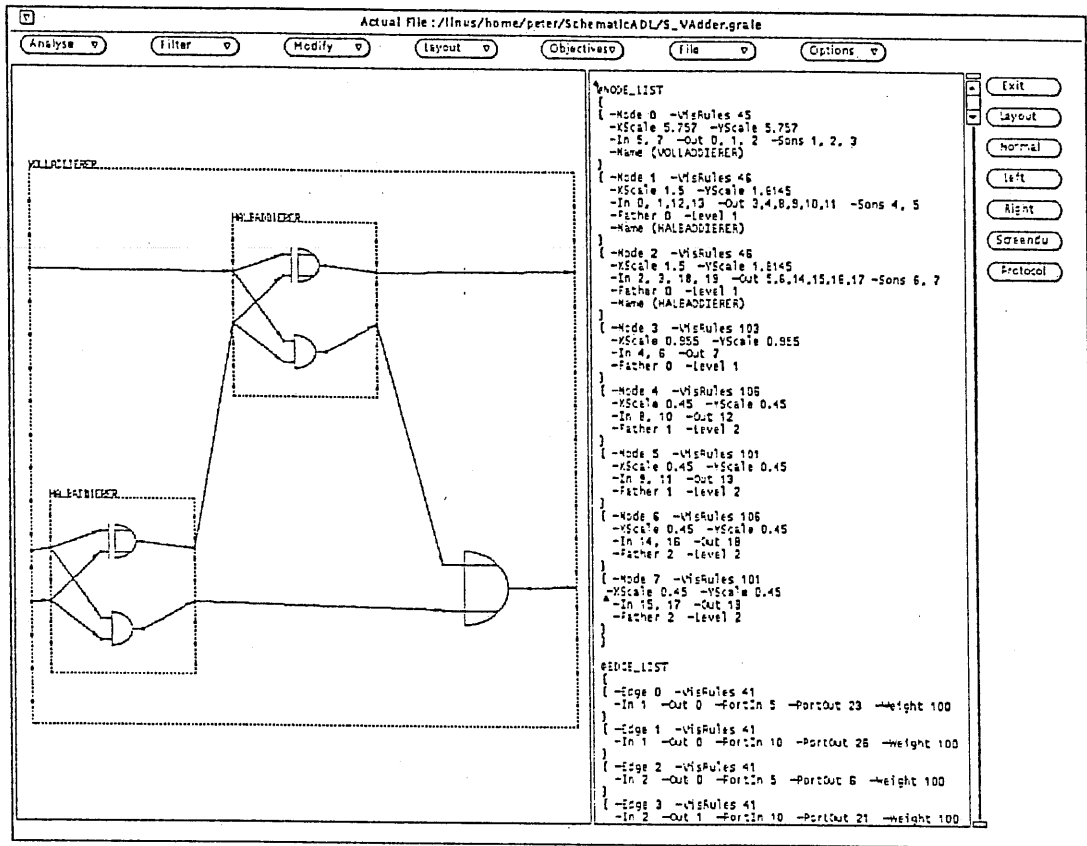


Figure 2: Screenshot of system *SchematicADL*.

since we still do not have much experience with these forces and since they are of no importance for the mode of operation of the algorithm itself, we postpone a detailed description to a later publication.

#### 4.4. Applications of Automatic Display Layout

In order to gain experience with algorithms and objective functions for “good” layouts, we have implemented two very different applications, graph layout in a window and display layout of several text windows. Both applications rely on the base tool *ScreenBroker*, an experimental tool for display layout. Its structure is discussed in detail in [14].

The tool *SchematicADL* generates automatic layouts of hierarchical circuit schematics. With special filter macros, like for searching the longest path, the designer is able to flexibly select the momentarily relevant data to display. A screen dump of the system *SchematicADL* is shown in figure 2. To achieve the typical left-to-right signal flow in schematics, the component DeltaX of the objective function is highly weighted.

It is obvious that for automatic generation of schematics the implementation of an algorithm for routing edges is necessary which is currently under development. But, as is easily seen from figure 2, the placement suits the typical “look and feel” of circuit schematics and routing would be easy.

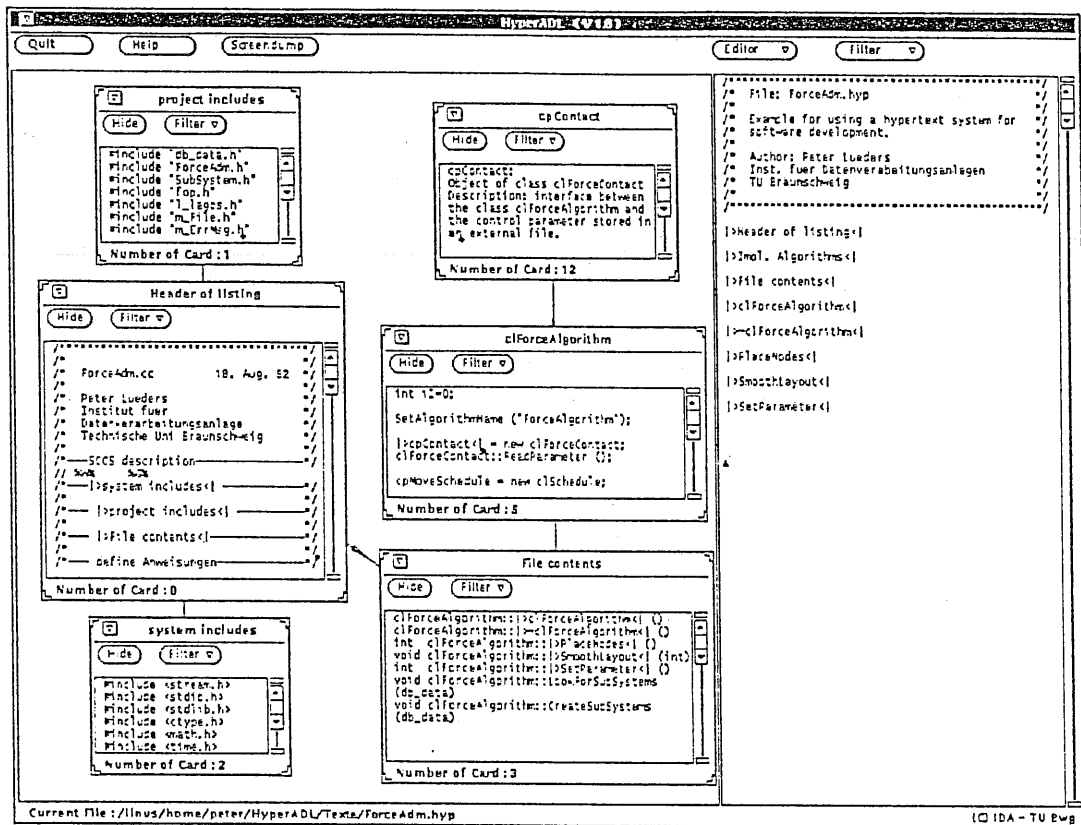


Figure 3: Screenshot of the hypertext system *HyperADL*.

The second application is *HyperADL*, a tool for writing / editing and reading hypertexts. Since we focus on the investigation of algorithms for display layout, the functionality of this tool is quite simple compared to other hypertext systems [17]. At system start the user gets two windows, one with a usual texteditor as a hypertext root window on the right side and, at the left, a second one for automatically placing the text cards. Loading a hypertext, the root card is displayed in the right window. Links are marked with the signs '[|>](#)' and '[<|](#)', respectively. Clicking the mouse button while pointing on the text between the marked signs will open the assigned card and display it with its own links on the left window. Using the mouse again the user is able to follow new links. Simple filters are also implemented. For example, it is possible to mark a card as actual and then all cards are shown which are referred by links starting from the actual card. A "least recently referred" filter can remove less interesting windows or resize them to icons.

A screenshot of the system *HyperADL* with a loaded source code is shown in figure 3. While the textual, compressed structure of the root card is shown in the right window, the left window displays some cards to the user. In the example, the cards on the left show the header of the source file with links to the include files and one function, again with a link to a variable description. Window placement, sizes and edge connections are generated fully automatic.

## 5. Conclusion

As a major step towards an improved user interface of an information processing system, we identified automatic display layout for graphical data representation and window placement. In order to investigate layout algorithms and to gain experience with the requirements for automatic display layout in an interactive environment, we implemented the experimental system, *ScreenBroker*.

The implemented layout algorithms are simulated annealing and genetic algorithm for a global layout, a look-up table approach for a local layout and a force directed algorithm for a postprocessing phase. They are general in nature and rely on an application specific objective function which evaluates a layout. We defined a weighted sum of objective function components to achieve a clearly arranged display. The components can be classified in two dimensions, namely the static and the dynamic dimension. While the static dimension is responsible for the intelligibility of a single layout, the dynamic dimension values the intelligibility of a series of displays.

The profit of computing the layout in three phases is the short computation time. The user can already learn the display at the end of the global layout while post processing still continues. The benefit of the introduction of an explicit application specific objective function is the flexibility of the *ScreenBroker*. Controlling the set of parameters for the objective function, various characteristics including the consideration of the topological consistency of the generated layout can be ensured. The power of the approach is shown by the implementation of two example systems, *HyperADL* and *SchematicADL*, each using its own objective function.

## 6. References

- 1 P. Eades and R. Tamassia, "Algorithms for Drawing Graphs: An Annotated Bibliography", Tech. Rep. CS-89-08, Department of Computer Science, Brown University, 1989.
- 2 F. Newberry-Paulisch and W. F. Tichy, "EDGE: An Extendible Graph Editor", *Software — Practice and Experience*, vol. 20, pp. 63-88, 1990.
- 3 L. A. Rowe, M. Davis, E. Messinger, C. Meyer, C. Spirakis, and A. Tuan, "A Browser for directed Graphs", *Software — Practice and Experience*, vol. 17, no. 1, pp. 61-76, 1987.
- 4 G. Robins, "The ISI Grapher: A Portable Tool For Displaying Graphs Pictorially", *ISI Reprint Series*, 1987.
- 5 G. Bruns, "GERM: A Metasystem for Browsing and Editing", tech. rep., MCC/Software Technology Program, 1988.
- 6 K. Sugiyama, S. Tagawa, and M. Toda, "Methods for Visual Understanding of Hierarchical System Structures", *IEEE Trans. on Systems, Man and Cybernetics*, vol. 11, no. 2, pp. 109-125, 1989.

- 7 B. Shneiderman and G. Kearsley, "Hypertext, Hands On!", *Addison Wesley*, 1989.
- 8 H. Langendörfer and M. Hofmann, "Das Hypertextsystem CONCORDE", *Informatik — Bericht 89-06, Technische Universität Braunschweig*, 1989.
- 9 E. S. Kuh and T. Ohtsuki, "Recent Advances in VLSI-Layout", *Proceeding of the IEEE*, vol. 78, no. 2, pp. 237-263, 1990.
- 10 D. E. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning", *Addison Wesley Publishing*, 1989.
- 11 J. P. Cahoon and W. D. Paris, "Genetic Placement", *Proc. IEEE Int. Conference on Computer Aided Design*, 1986.
- 12 S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing", *Science*, vol. 220, no. 4598, pp. 671-680, 1983.
- 13 R. Otten and L. van Ginneken, *The Annealing Algorithm*. Kluwer Academic Press, 1989.
- 14 P. Lüders and R. Ernst, "Verbesserung der Benutzeroberfläche von CAD-Systemen durch automatisches Bildschirmlayout (Improvement of the User Interface by Automatic Window Layout)", *CAD 92, Springer*, p. 77 — 93, 1992.
- 15 P. Eades, "A Heuristic for Graph Drawing", *Congressus Numerantium*, vol. 42, pp. 149-160, 1984.
- 16 R. Tamassia, G. di Batista, and C. Batini, "Automatic Graph Drawing and Readability of Diagrams", *IEEE Trans. on Systems, Man and Cybernetics*, vol. 18, no. 1, pp. 61-79, 1988.
- 17 R. Bogaschewsky, "Hypertext- / Hypermediasysteme — Ein Überblick", *Informatik Spektrum*, vol. 15, no. 3, pp. 127-143, 1992.