Multimedia and Networking Conference, San Jose, California, 6.-8. Feburar 1995

Improvement of the User Interface of Multimedia Applications by Automatic Display Layout

Peter Lüders and Rolf Ernst

Technische Universität Braunschweig, Institut für
Datenverarbeitungsanlagen, Germany, D-38106 Braunschweig

ABSTRACT

Multimedia research has mainly focussed on real-time data capturing and display combined with compression, storage and transmission of these data. However, there is another problem considering real-time selecting and arranging a possibly large amount of data from multiple media on the computer screen together with textual and graphical data of regular software. This problem has already been known from complex software systems, such as CASE and hypertext, and will even be aggravated in multimedia systems. The aim of our work is to alleviate the user from the burden of continuously selecting, placing and sizing windows and their contents, but without introducing solutions limited to only few applications. We present an experimental system which controls the computer screen contents and layouts, directed by a user and/or tool provided information filter and prioritization. To be application independent, the screen layout is based on general layout optimization algorithms adapted from the VLSI layout which are controlled by application specific objective functions. In this paper, we discuss the problems of a comprehensible screen layout including the stability of optical information in time, the information filtering, the layout algorithms and the adaptation of the objective function to include a specific application. We give some examples of different standard applications with layout problems ranging from hierarchical graph layout to window layout. The results show that the automatic tool independent display layout will be possible in a real time interactive environment.

Keywords: User interfaces, multimedia, information retrieval, combinatorial optimization, graph layout

1. INTRODUCTION- THE PROBLEM OF DISPLAY LAYOUT

Today, more and more computing systems are connected through local and/or wide area networks. Terms like cooperative processing, multimedia or electronic publishing describe the complex interactions between individuals or groups of users. State of the art in software systems are programs which often use several windows in a window oriented user interface to display or to ask for information. The user faces a bulk of information and has to learn the structure and contents of the actual display while trying to get orientation in the network of information itself at the same time.

Therefore, the problem of handling the size and the locations of the different windows becomes highlighted. Experiences with complex software systems show that the user will spend a lot of time with sizing and moving the windows to get the currently interesting information. For these reasons we regard the introduction of display layout - automatically sizing and placing the displayed windows on the screen - as an essential improvement for the user interface.

Within next generation multimedia applications and highly interactive systems a typical user-interaction scene could look like the one sketched in figure 1. The user controls the displayed information by individual information filter, either quite simple ones like adding / deleting one window or even more complex ones like retrieving connected information from a large (possible distributed) data base. Once the information to display is specified, layout algorithms compute a new layout and display it onto the screen.

The problem, however, is to place the elements of a hierarchical graph, given by a set of nodes and a set of edges (directed and/or undirected) on a given placement area. In our approach, a hierarchical structure is considered with the introduction of parent-child relations and the hierarchy is graphically represented by placing the children onto the parent node.

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

- The total amount of objects which could be displayed on the display is less than 50 for a reasonable display layout.

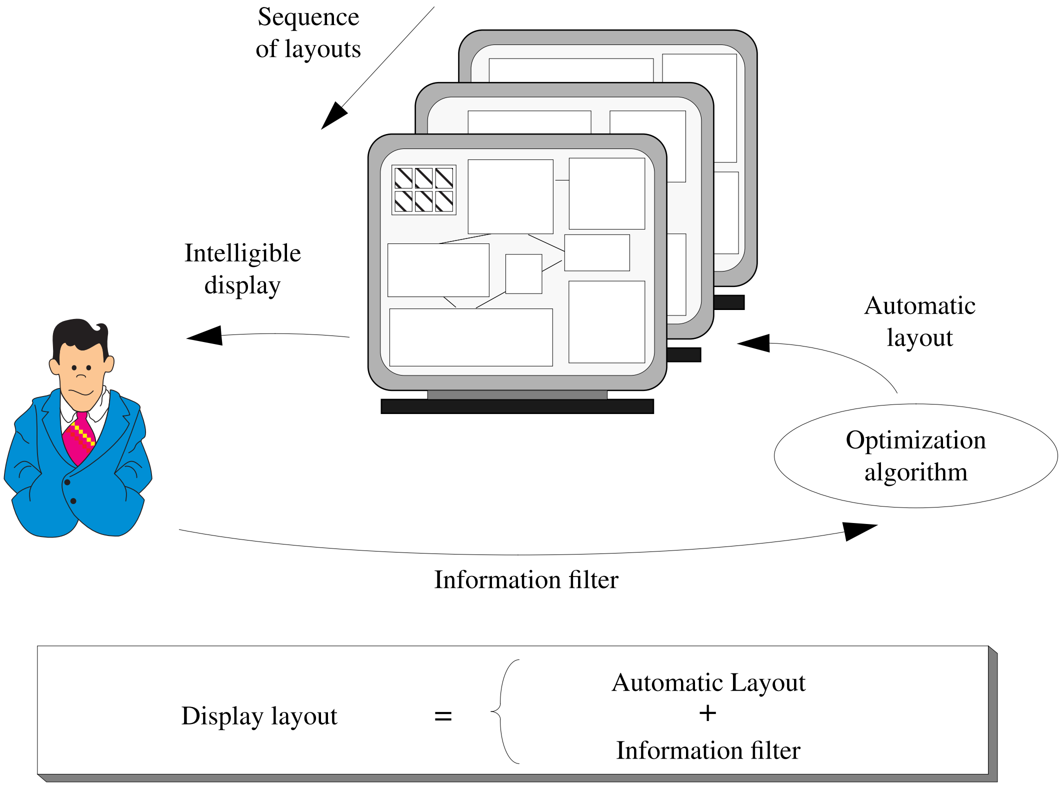


Figure 1: Connectivity between user, information filter and layout algorithms.

- The size of each node (window) could range between iconization and full screen mode.
- Different users and/or different applications prefer various kinds of representation (example: schematic versus hypertext).
- The response time to a user request has to be in the range of a second to permit an efficient work.
- To ensure continuity during a user‘s work, small changes in the display structure - like opening or closing a single window - have to lead to a new display with a similar layout topology. It is unacceptable for a user to have to deal with a completely new layout subsequent to each layout computation. We have introduced the term *topological consistency* for this requirement.

While 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 us to split up the computation of a display layout into two parts that are *global layout* and *postprocessing*. The essential benefit of using combinatorial optimization algorithms is the possibility of controlling 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 optimize its value during the computation. Because the algorithms optimizes the objective function, modifying the objective function results in different layouts emphasizing the graph characteristics evaluated within it. The introduction of the two-phase-approach leads to small calculation times for layout computation and thereby ensures interactive work.

2. PREVIOUS AND RELATED WORK

In some systems for generating multimodal presentations like WIP[1,2], its successor PPP[3] or COMET[4] a knowledge based approach to compute the display layout has been worked out. Within these systems, constraints for the new layout are derived by large knowledge bases. Subsequently the layout is computed by a constraint solver which generates layouts with as most as possible constraints solved. But the computation times seem to be very large. Therefore the practical use of this approach seems not to be manageable in interactive environments, even regarding the computing power of next generation workstations. Furthermore, it is very difficult for a user to express desired layout characteristics by using the mechanisms of constraints only.

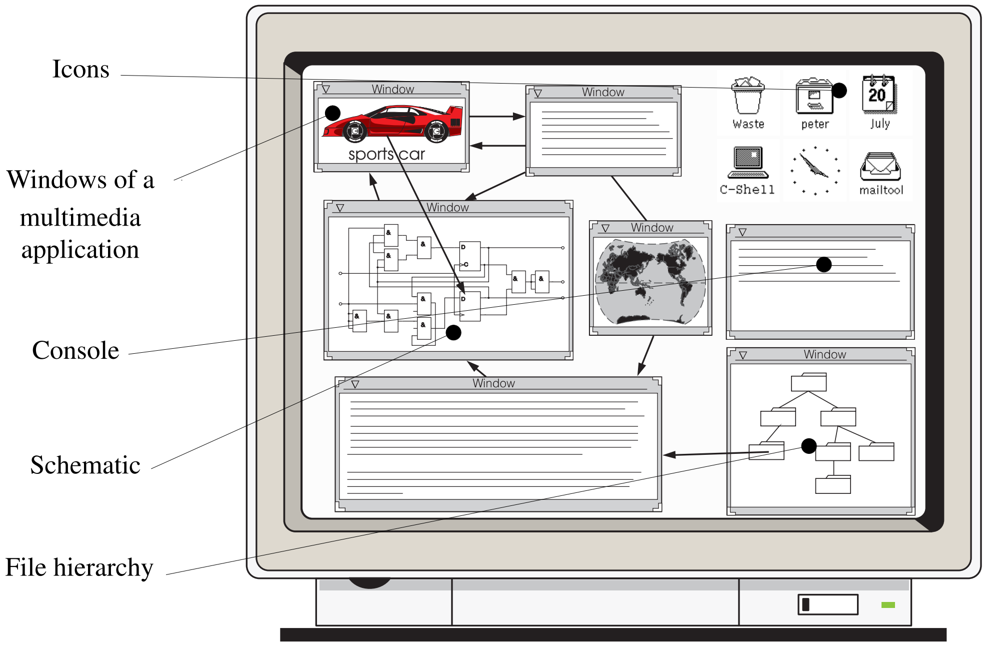


Figure 2: Introduction of the problem of display layout by an example.

A general bibliography for graph layout algorithms is already published[5]. Examples for browser are Bonsai[6] EDGE[7], GRAB[8], ISI Grapher[9], GERM[10] and GraphED[11]. Their placement algorithms are based on an algorithm first described by Sugiyama et. al.[12]. However, 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 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, it is regarded in EDGE 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 define explicitly an objective function to control directly the algorithms with regard to topological consistency.

In some Hypertext systems[13] automatic display layout is implemented to visualize the structure of the hypertext. Here, simple deterministic algorithms are used to place equal sized nodes based on preset heuristics. Again, the problem of topological consistency is not taken into consideration. But in addition to browsers, hypertext systems improve the user interaction by possibilities to navigate through the graph and by filter macros to select specific information.

A general survey of the VLSI layout is published by Kuh and Ohtsuki[14]. The main objective of placement algorithms used in VLSI layout is to achieve routability subsequent to the placement phase and the maintenance of given circuit characteristics. User interactions during the computation or regarding the topological consistency in subsequent layouts are not taken into consideration.

The general structure of the experimental tool GraphADL and the examination of the implemented algorithms are published in a previous paper[16]. The implementation of layout algorithms relies on an implementation developed by Otten and Ginneken[15] (simulated annealing) and on previous work by Eades[17] and by Fruchterman and Reingold[18] (force directed placement). The basic attributes of a graph evaluated in the objective function to direct the optimization algorithms are based on the constraints for graph display given in[19].

3. APPLICATIONS FOR AUTOMATIC DISPLAY LAYOUT

For the investigation of layout algorithms and the requirements of a system for independently sizing and placing windows in a window oriented user interface, we implemented three applications which span the gamut between display layout and hierarchical graph layout.

3.1. Editor for Hypertexts: HyperADL

HyperADL is a tool for editing and reading hypertexts. Because we focus on the examination of algorithms for display layout, the functionality of this tool is quite simply compared to other hypertext systems. Loading a hypertext, the root card is displayed in the center of the main window. Links are marked with the signs '|>' and '<|', respectively. The clicking of 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 could follow new links. Superfluous windows are closed by clicking the „hide"-button on the frame of the card. Simple filters are also implemented. For example, it is possible to look for all cards links within the actual card points to or to open all cards with links to the actual card. For using HyperADL as a source file editor, the root textual window has a special filter called „combine" to linearize the total source code for compiling.

An example for a sequence of layouts while working with the tool *HyperADL* is shown in figure 4a - 4e. The first screendump shows the screen after loading a hypertext. Only the root card of the hypertext is shown in the center of the screen (a). Subsequently the user looks for all cards which could be reached by the root card (b). Then he hides two cards (c) and selects one card as being of interest resulting in a larger window size (d). After reading the card the user selects another card, resulting again in a larger window size (e).

3.2. Window Manager with Automatic Display Layout: WinManADL

The system WinManADL controls the layout of the whole display, i.e. all individual windows of the display. The user

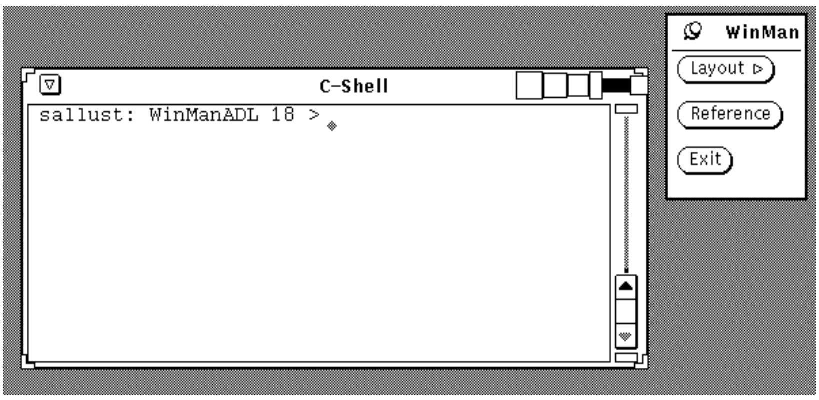
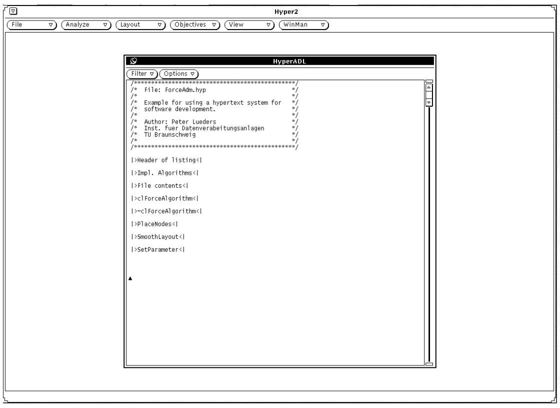


Figure 3: New frame decoration elements for setting specific attributes (desired size and desired width to length ration) of a window within the window manager WinManADL.

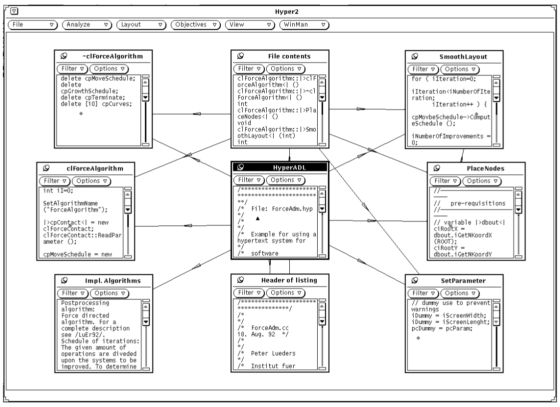
assigns specific attributes to each window which are regarded within the next layout generation. In a first attempt we added six carets within the frame of each individual window (see figure 3). Using the left boxes, the user assigns a new desired size, using the right boxes, the user assigns a new desired width to length ratio. More specific attributes could be entered by a special pop up panel.

3.3. Browser for General Graphs: GraphADL

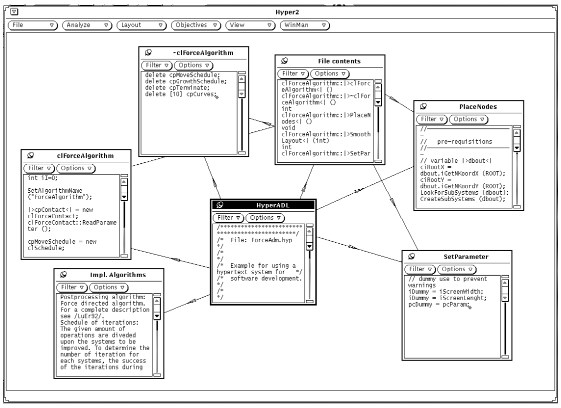
GraphADL (ADL: *A*utomatic *D*isplay *L*ayout) is a browser for general graphs and is mainly used for the developing and testing of the implemented layout algorithms and as a root system within the object oriented software hierarchy.



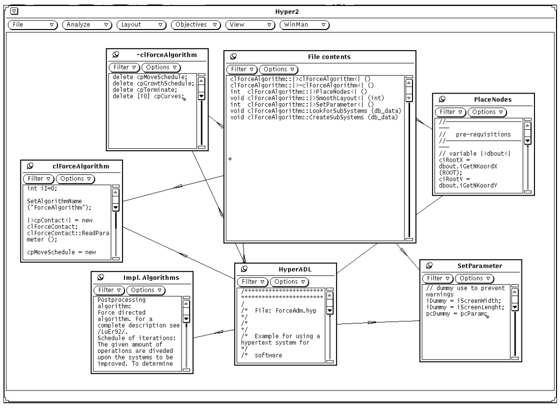
a) Loading a hypertext.



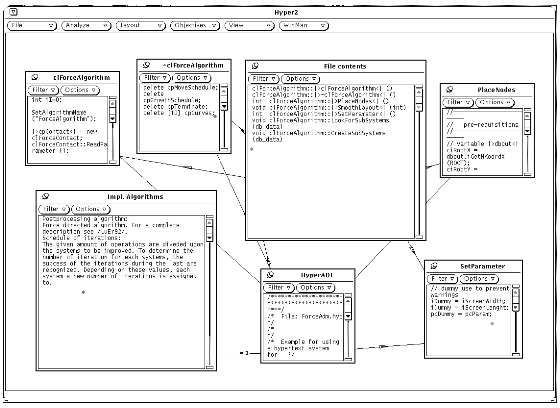
b) Displaying all reachable cards.



c) Hiding two cards.



d) Selecting a specific card.



e) Selecting an additional card.

Figure 4: A sequence of display layouts while working with the hypertext editor *HyperADL*.

Once the user has started GraphADL, configuration parameters like the last file worked with are read. Further, a textual description of possible forms or shapes for the graphical representation of nodes and edges is read. Using the formal language *lagos* (*la*nguage for *g*raphic *o*bject*s*), most shapes for nodes and edges needed in various applications from schematic to hypertext could be designed. At the end of the initialization, the last file worked with is loaded and then displayed.

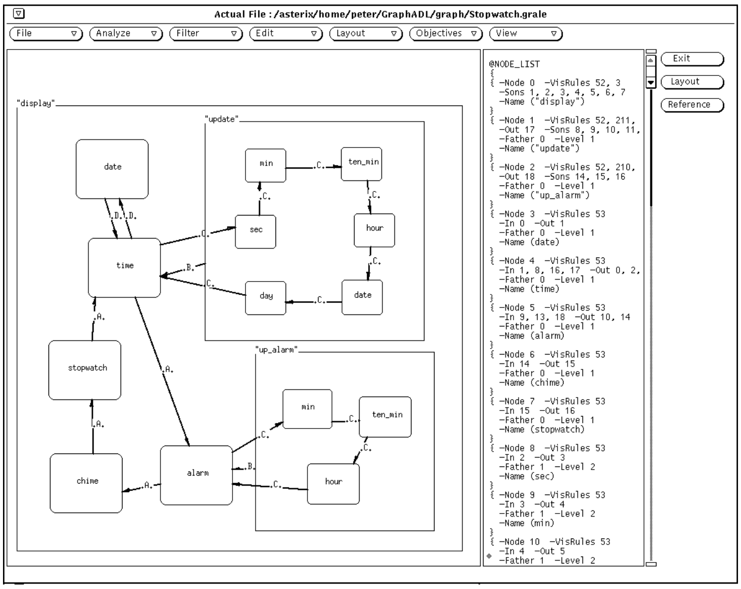


Figure 5: Screendump of system GraphADL.

Now, the user could edit the textual description or execute analyze, modify or filter macros. Using analyze macros, specific information within the display could be highlighted, for example by changing color or shape. Although a new display has to be created in order to regard the changed attributes, computation of a new layout is not necessary. Modify macros give the user an easy way to edit the graph data. Graph elements could be added or deleted with these macros. Editing the graph with modify macros has the benefit to ensure data consistency because all work to delete or add an element is done by the macro during its execution. A reducing of the amount of information displayed can be achieved by the execution of filter macros. They are useful while working with large graphs. The user could select specific information for which a new display is computed for. Typical implemented filter macros which are useful for various applications are:

- specifying a set of nodes and a degree of neighborhood: the chosen nodes with all nodes which could be reached using a number of edges smaller than the given degree are selected. Note, that the direction of the edges is disregarded by this filter.
- same as first, but with regard to the direction of the edges.
- same as first, but with edges instead of nodes.
- specifying a node: the chosen node and all nodes placed within it are selected.
- specifying a set of nodes: all paths between the chosen nodes are determined and each path is represented by a single edge.

Within the execution of a macro, the graphical representation of a graph element has to be altered or has to be set. This is done by adding visualization rules, which set for each element the attributes for display like color, form or size. Before starting the layout algorithm, parameters to control the computation of the layout and the type of objective function are determined depending on the actual type of graph (directed, undirected graph) and on the actual work of the user (incremental change, completely new layout). At last, a new display layout is computed and displayed on the screen.

3.4. Schematic editor: SchematicADL

The system SchematicADL generates automatic layouts of schematics based on a given netlist or formal textual description of the nets. This allows a familiar sight of synthesized circuits or circuits described in a textual language (example: VHDL). With special filter macros like searching the longest path the designer could select the currently relevant data to concentrate on. A screendump of the system SchematicADL is shown in figure 6. It has to be mentioned that the computation of the rectilinear lines is done again by a two phase routing approach, namely simulated annealing optimization algorithm for the global routing phase followed by a heuristic algorithm for the local phase.

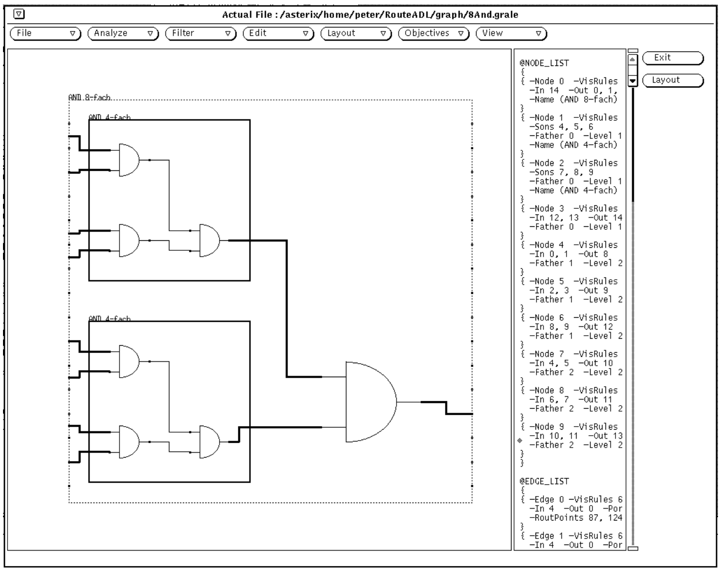


Figure 6: Screendump of system SchematicADL.

4. COMPUTATION OF DISPLAY LAYOUT BY COMBINATORIAL OPTIMIZATION ALGORITHMS

The profit of using combinatorial algorithms for display layout is the separation between the cost function which evaluates a solution as ‘good‘ or ‘bad‘ and the strategy of finding a ‘good‘ solution based on the actual cost function. In our approach, we have defined several functions to evaluate obvious characteristics of a good layout which will be summed up to the objective function.

The computation of a new display layout starts with the definition of the window structure, especially the contexts between the windows and the desired size of each window for its appearance on the screen. This could be done by the user or by the application. The task of the first layout phase, the global layout phase, is to compute a coarse raster based layout regarding the actual cost function. To avoid overlappings, the size of the windows is adjusted to the raster. We use the simulated annealing optimization algorithm for this task. This algorithm is intensively used within VLSI chip layout which is a similar problem to display layout.

But because of the coarse raster and the adjusted node sizes, the result could be characterized by orthogonality and inefficient use of the screen area. During the postprocessing phase the size of the nodes are iteratively increased or decreased considering the desired size for each node. To avoid window overlappings, the nodes are also moved locally. These improvements are iterated until a given time limit or limit of layout quality is exceeded. For this task an enhanced force directed placement algorithm is used.

4.1. Global Layout Phase

4.1.1. Objective Function

An essential task for computing the size and placement of windows in a window oriented user interface or a graph layout within a single window is the examination and implementation of an objective function to evaluate the quality of a computed layout[19]. Because the optimization algorithms improve a layout relying on the evaluation of the objective function, it is decisive to get a intelligibly arranged layout. In our approach we have defined in a first step several functions to evaluate obvious characteristics of a good layout which will be summed up to the objective function.

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

- *Edge Length*: as in most approaches for computing graph displays, we define the minimization of the total edge length as an optimization criterion. This results in placing connected nodes together and avoids large edges producing badly arranged displays.
- *Homogeneous distribution*: agglomeration of nodes in several parts of the placement area seems to be undesirable, therefore the homogeneous distribution of the nodes on the display size will be an attribute of a clearly arranged layout, too.
- *Edge crossing*: the number of edge crossings has to be minimized.
- *Visibility*: in the current implementation, overlappings of nodes are not allowed to ensure the visibility of edges connecting the nodes and the contents of each node.
- *Edge direction*: for directed graphs like flow charts or schematics the direction of the edges also has to be evaluated. Edges going straight left are regarded as well oriented edges.
- *DeltaX*: for directed graphs again, the flow control is evaluated by looking for the difference of the x-coordinates of two connected nodes. If there is an edge going from node 1 to node 2, the x-coordinate of node 2 has to be larger than of node 1. Otherwise, the placement is badly evaluated.

To quantify the dynamic part of the objective function, we have implemented two elements:

- *Absolute changes*: the sum of the distances of all nodes between their new and old locations gives the first element for regarding the topological consistency. This results in nearby locations of the nodes in subsequent figures.
- *Relative position*: the changes in the relative positions between the nodes are evaluated, too. This attribute can be valuated by the difference in the directions of imaginary lines connecting the centres of all nodes in subsequent displays.

The complexity of the objective function is essential for the computation time. For a graph with approximately 10 - 20 nodes it is called several thousand times during the computation process. Therefore it is necessary to be very careful in designing the objective function. In our practical work, we use mainly the static attributes of edge length (and DeltaX) and the dynamic attribute of absolute changes to compute a layout. These attributes could be calculated easily because 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 found out that with minimizing the total edge length the number of edge crossings tends to decrease, therefore the calculation of the edge length seems to be obvious.

4.1.2. The Simulated Annealing Optimization Algorithm

The task of the global layout phase is to compute a raster based layout which is clearly arranged considering the given objective function. 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 the adjusted raster for possible locations of nodes. At first, the objective function could be simplified because homogeneous distribution is ensured and an overlapping of nodes can not occur. Secondly, temperature and annealing function of the simulated annealing depends on the possibilities of moving a node. Reducing these possibilities results in faster annealing at lower temperatures and therefore less computation time.

The implementation of the adaptive simulated annealing is based on a description by Otten and Ginneken[15]. 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. 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.

4.2. The Postprocessing Phase

The display layout computed during the global layout phase could be characterized by its orthogonality and its inefficient use of the placement area, especially for hierarchical displays. The main task of the postprocessing algorithm is to improve these attributes.

In our approach, we use a force directed algorithm to improve the layout quality. Within this algorithm, the edges of the graph are mapped to strings which are set 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 Eades[17] and Fruchterman and Reingold[18]. Here, attractive and repulsive forces (to prevent a contraction into one point) are used to compute a graph layout.

To adapt this algorithm to the hierarchical display layout, several new forces have to be introduced. The repulsive and the attractive force listed are the traditional ones for graph layout. The border force has to be defined to regard the hierarchical structure of the graph. Furthermore, the relative and the absolute displacement forces are introduced to consider the topological consistency.

- *Repulsive force*: between each pair of nodes a force is defined for pushing the nodes apart for a homogeneous distribution. Its strength is inverse to 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 would be placed in the centre of their placement area. Like the repulsive force, its strength is inverse to the distance to prevent to move a node out of its placement area.
- *Relative displacement force*: forces depending on the relative displacements of two nodes between the actual layout and the previous layout.
- *Absolute displacement force*: forces depending on the absolute displacement of each object between the actual layout and the previous layout.

The orthogonality of the layout could be improved by the force directed algorithm using the named forces. Considering the problem of altering node sizes for an efficient use of the placement area, which could result in enlarging and / or reducing some nodes or in reducing some nodes to enlarge others, additional functionality has to be introduced. 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 by:

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

The algorithm starts with the calculation of forces and pressures effected on each node during the initialization phase. Within the main iteration loop, the system is improved at first by moving a node and secondly by altering the size of a node. The node to be moved is determined by searching for the node with the largest force effected on. Once it is found, an incremental move in order to lessen the force is executed. The length of the move is given by a user defined parameter adjusted to the size of the actual system. In a similar mode of operation the node to alter its size is found. Here, the result of the substraction of outside pressure and inside pressure is calculated for all nodes. The node with the maximal sum is searched and depending on the sign of the result its size is enlarged or reduced. Again, the factor for changing the size is given by the user. It is to be mentioned, that at the current state of work, the enlargement of node sizes by the same factor for both directions is the only possibility of altering node sizes. The main iteration stops when a given amount of iterations is reached or the improvement between subsequent iterations remains under a given limit.

One profit of the force directed algorithm is the possibility to show the computation of a layout directly on the screen. Because of the incremental modifications within each iteration the user gets the impression of a continuous flow of operation. An example with several screendumps of a layout computation is presented in figure 7.

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. To investigate layout algorithms and to get experience with the requirements of automatic layout in an interactive environment, we implemented the experimental systems GraphADL, SchematicADL, HyperADL and WinManADL.

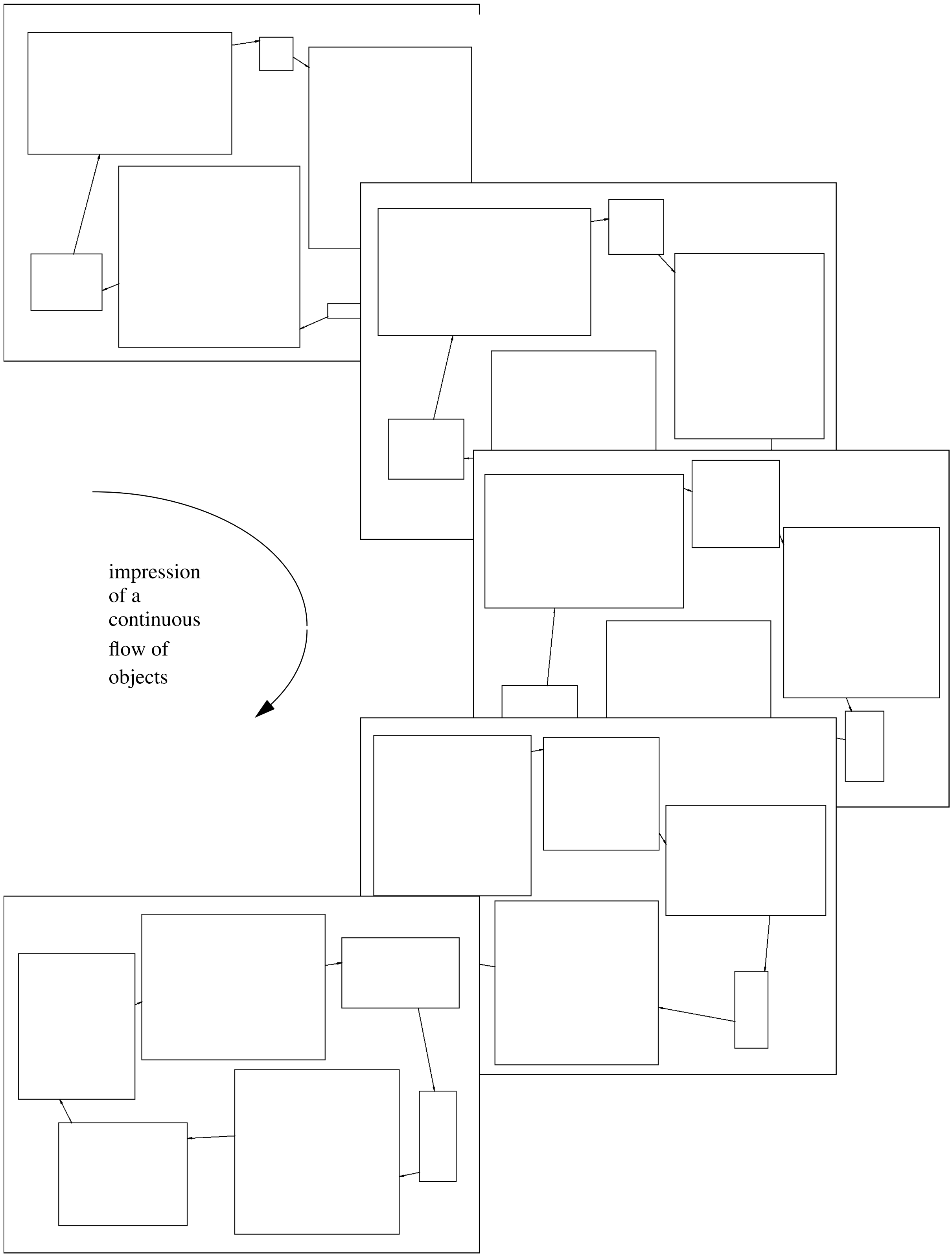


Figure 7: Redrawing the temporary layout within each iteration of the force directed algorithm results in the impression of a *continuous flow of objects* (top left to bottom right).

We present our approach to compute a display layout in two phases that are global layout and postprocessing. At first a display layout based on a coarse raster is computed and displayed on the screen to give the user a first impression of the new layout at the end of the global layout phase. Because of the raster, the layout is orthogonal and the screen size is inefficiently used. These disadvantages are removed during the postprocessing phase. The layout is iteratively improved with incremental changes, which are displayed on the screen to give the user the impression of a continuous flow of objects.

The implemented layout algorithms are the simulated annealing algorithm for global layout and the force directed algorithm for the postprocessing phase. They rely on an objective function which evaluates a given layout. We have defined a set of obvious elements of the objective function for evaluating selected characteristics of a clearly arranged display. The elements can be classified as two dimensions namely the static and the dynamic dimension. While the static dimension is responsible for the intelligibility of a single layout, the dynamic part values the intelligibility of subsequent displays.

The profit of computing the layout in two phases is the short computation time until the first display of a layout is shown on the screen. The user can already learn the display at the end of the global layout while the computation still continues. The benefit of the introduction of an objective function is flexibility and tool independency. The flexibility is shown by the implementation of the demonstration systems, each using its own type of objective function. Computing times are within the range of seconds for typical screen contents (about 20 objects) on a SparcStation 20; these data show that real-time execution is possible in next generation‘s personal computers.

6. References

1 E. André, W. Finkler, W. Graf, T. Rist, A. Schauder, W. Wahlster: WIP: The Automatic Synthesis of Multimodal Presentations, Research Report 92-46, DFKI, Saarbrücken, Germany, 1992.

2 W. Graf: Constraint-Based Graphical Layout of Multimodal Presentations, Research Report RR-92-15, DFKI, Saarbrücken, Germany, 1992.

3 E. André, W. Graf, T. Rist, J. Heinsohn, B. Nebel, H. Profitlich, T. Rist und W. Wahlster: PPP: Personalized Plan-Based Presenter, Research Report 93-05, DFKI, Saarbrücken, Germany, 1993.

4 S. Feiner, K.R. McKeown: Automating the Generation of Coordinated Multimedia Explanations, IEEE Computer, Oktober, 33-41, 1991.

5 Giuseppe Di Battista, Peter Eades, Roberto Tamassia und Ioannis G. Tollis: Algorithms for Drawing Graphs: an Annotated Bibliography, anonymous ftp from wlima.cs.brown.edu, 1993

6 J. I. Helfman: Bonsai: A Prototype Graph Browser and Graphical Interface Tool, AT&T Bell Laboratories, CMP Computing, 1987

7 Frances Newberry-Paulisch und Walter F. Tichy: EDGE: An Extendible Graph Editor, Software - Practice and Experience, 20, 63-88, 1990

8 Lawrence A. Rowe und Michael Davis and Eli Messinger and Carl Meyer and Charles Spirakis and Allen Tuan: A Browser for directed Graphs, Software - Practice and Experience, 17, 1, 61-76, 1987

9 Gabriel Robins: The ISI Grapher: A Portable Tool For Displaying Graphs Pictorially, ISI Reprint Series, 1987

10 Glenn Bruns: GERM: A Metasystem for Browsing and Editing, MCC/Software Technology Program, 1988

11 Michael Hinmsolt: Graph^{ED} User Manual, Universität Passau, 1990

12 Kozo Sugiyama und Shojiro Tagawa and Mitsuhiko Toda: Methods for Visual Understanding of Hierarchical System Structures, IEEE Trans. on Systems, Man and Cybernetics, 11, 2, 109-125, 1989

13 Ben Shneiderman und Greg Kearsley: Hypertext, Hands On!, Addison Wesley, 1989

14 Ernest S. Kuh and Tatsuo Ohtsuki: Recent Advances in VLSI-Layout, Proceedings of the IEEE, 78, 2, 37-263, 1990

15 R.H.J.M. Otten und L.P.P.P. van Ginneken: The Annealing Algorithm, Kluwer Academic Press, 1989

16 Peter Lüders and Rolf Ernst: Automatic Display Layout in Window Oriented User Interfaces, IFIP Workshop on Interfaces in Industrial Systems for Production and Engineering, Darmstadt, 1993

17 Peter Eades: A Heuristic for Graph Drawing, Congressus Numerantium, 42, 149-160, 1984

18 Thomas M.J. Fruchtermann and Edward M. Reingold: Graph Drawing by Force-directed Placement, Software - Practice and Experience, Vol 21(11), 1129-1164, 1991

19 Roberto Tamassia and Guiseppe di Batista and Carlo Batini: Automatic Graph Drawing and Readability of Diagrams, IEEE Trans. on Systems, Man and Cybernetics, 18, 1, 61-79, 1988