13th IFIP World Computer Congress 1994 (Vol. I), 28.Aug. - 2.Sept., Hamburg, Deutschland, 1994

The Dynamic Screen - Beyond the Limits of Traditional Graphic User Interfaces

P. Lüders, R. Ernst

Technische Universität Braunschweig, Institut für Datenverarbeitungsanlagen,
Hans-Sommer-Straße 66, 38106 Braunschweig, Germany

Keyword Codes:H.5; H.3.3; G.1.6
Keywords: Information Interfaces, Information Retrieval, Optimization

1. Introduction

The small screen area is one essential limit for the amount of displayable information and for the amount of user receptable information in computer assisted visualisation and animation of complex information. During the last few years this limit has been shifted with the advent of high resolution graphic displays and with the introduction of the window oriented user interfaces. But it is reached again by actual or even more by future complex software systems like hypertext or multimedia systems. These systems can be characterised on the one hand by more and more user interaction and on the other hand by handling more and more complex information, which has to be displayed onto the screen.

For both user interaction and information display additional windows are used extensively. Therefore, many windows appear and disappear from the screen during the work with these systems. Therefore, the user has to manage the display and to find out the contexts between many different windows for understanding the displayed information. This can be a difficult and time consuming task because the windows appear at preset locations without regarding the actual window layout on the screen. In most cases, several windows overlap each other possibly hiding currently interesting information or worse, they hide each other completely.

An example should illustrate the mentioned limit. Let us assume that a user works with a hypertext system or a source code browser within object oriented software development. He wants to get some information about a specific class and subsequently take a look at related classes (furthermore we will use the term *information filter* for these actions). To solve this task, one or more windows appear for naming the specific class and the requested information about it. After the class name is entered, these windows disappear and other windows appear with information about this class. After reading this information, the user specifies the next request (information filter) within a new window and additional windows appear onto the screen showing the new information. In general, the context of the windows, i.e. the inheritance or other connections between the individual classes or windows is not taken into consideration for the location of the appearance of the new windows. The user has to arrange the window layout manually to get an intelligible information display .

For these reasons, we regard the introduction of display layout - automatically sizing and placing the displayed windows on the screen - in combination with dynamic information filtering as an essential improvement for the user interface. This fact could be summarized by the equation:

display layout = screen layout + information filter

Therefore, using automatic display layout, the limit of screen area is shifted again to permit the handling of more and more complex information display and user interaction.

2. Display Layout: An Optimization problem

The problem of display layout, 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. It seems as if the problem could be solved easily using traditional well known graph layout algorithms /2/. But several requirements have to be regarded within display layout which are not considered in graph layout approaches:

- The size of the nodes (windows) could range between iconization and full screen mode.
- Different users and / or different applications prefer various kinds of representation.
- The response time to a user request has to be in the range of a second to permit an efficient work.
- At last, the topology of the screen layout has to be similar in subsequent layouts. Otherwise, the user is confused by completely new layouts effected by small changes within the window structure. For this problem, we use the term *topological consistency* .

While the requirement of flexibility has led us to general combinatorial optimization algorithms /1/, the restriction of computation time has forced the splitting of the computation of a display layout in the global layout phase and the post-processing phase.

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. Examples for *static* components are total edge length, homogenous distribution of nodes or total number of edge crossings.

The problem of *topological consistency* is mentioned with specific objective functions regarding a previous layout set by the user as the reference layout. Evaluating the difference between the relative position of each pair of nodes within the actual and the reference layout is an example for a *dynamic* component. The outline of the system is shown in figure 1. Therefore, using different cost functions leads to different layout solutions, each based on the given cost function.

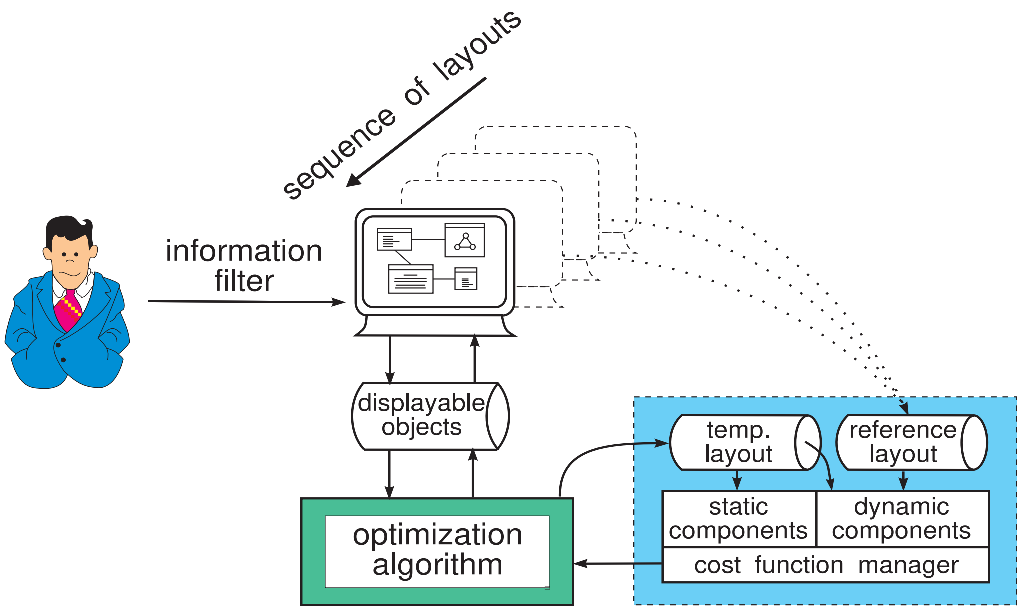

Figure 1. Outline of the computation of a display layout regarding a user preset reference layout by specific components of the cost function.

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 /5/ 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 characterised by orthogonality and inefficient use of the screen area. During the postprocessing phase the size of the nodes is iteratively increased or decreased depending on 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 /6/ is used. It has to be pointed out that the layout is changed incrementally only within each iteration which allows a direct display of the changes on the screen. This results in a *continuous flow of objects* for the user and improves the orientation of the user within the new layout. An example for

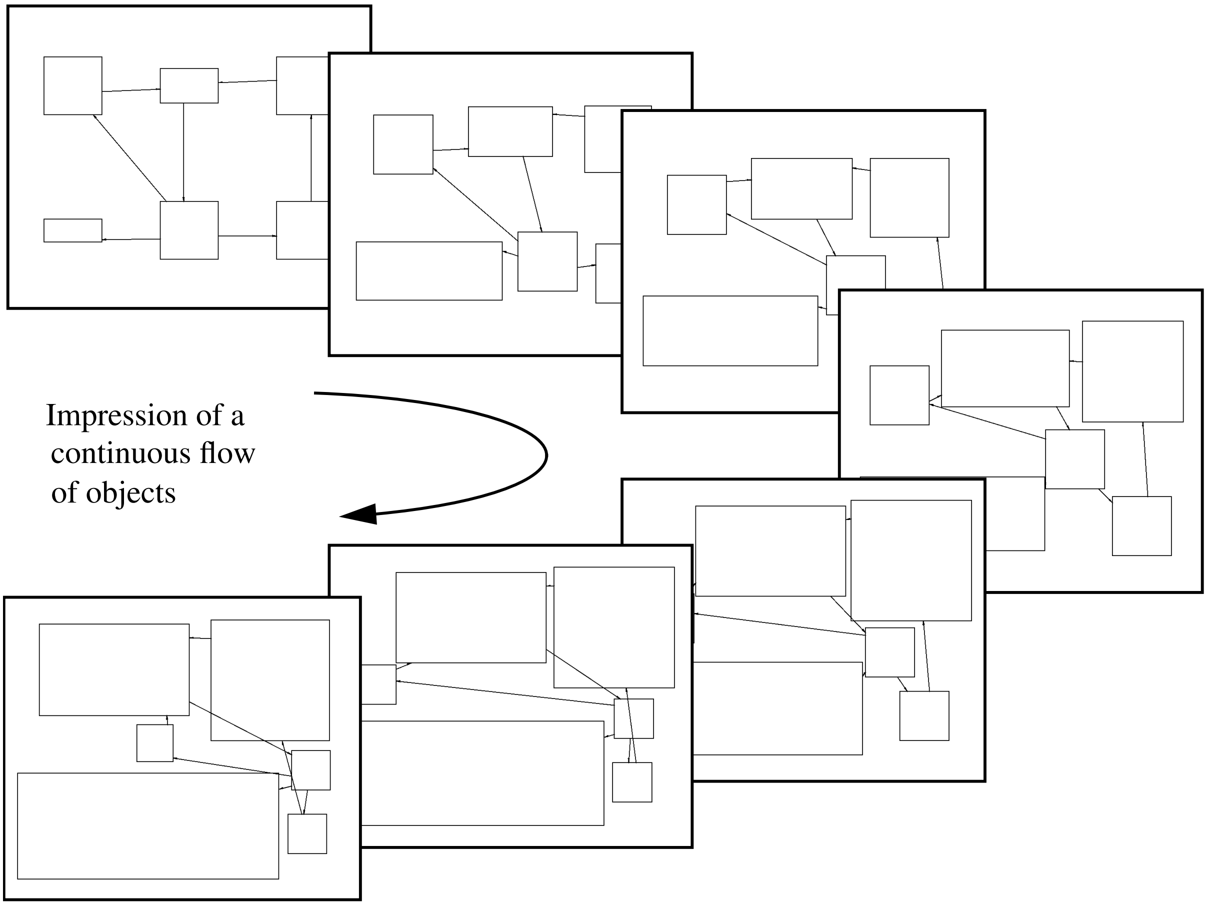

Figure 2. Redrawing the temporarily layout within each iteration of the force directed algorithm results in the impression of a *continuous flow of objects* (top left to bottom right).

the flow of objects is shown in figure 2. The raster based layout on the top left is improved by incremental changes up to the layout result on the bottom left. The rectangles itself stand for objects within an application using graph display or for individual windows within an window oriented graphic interface.

We have to split the layout computation into two parts to ensure a short computation time and therefore to permit interactive work. Within the first step the nodes are placed on a coarse raster (reduced complexity) regarding the given objective function. During the second step the raster is neglected but only local changes to the layout are allowed.

3. Experimental Tools for the Automatic Display Layout

In order to find out in more detail what the requirements of display layout are, we have implemented some applications which span the gamut between graph layout and display layout /3/, /4/. *GraphADL* is the experimental system to exam-

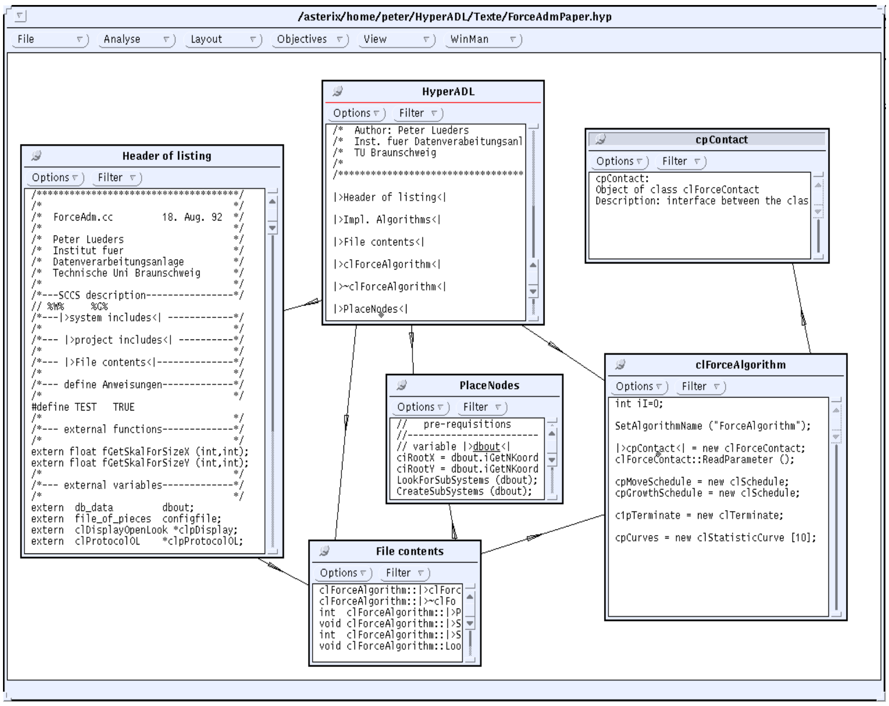

Figure 3. Screendump of the system HyperADL.

ine the layout algorithms. All other systems are inherited from it. As an individual system, it is used to visualize all types of graphs. The system *SchematicADL* generates automatic layouts of schematics based on a given netlist or formal textual description of the net. This allows a familiar sight of synthesized circuits or circuits described in a textual language (example: VHDL). With special macros for information filtering like searching the longest path the designer could select the momentarily relevant data to concentrate on. *HyperADL* is a simple system for entering and reading hypertexts (the screendump in figure 3 shows a possible layout while reading program code). Each card is represented by an individual window and the layout of the windows is automatically computed with each change of the information display. Further, the user could select specific information with information filter like seeing all cards which could be reached by the actual selected card. At last, *WinManADL* could be used on top of the usual window manager for automatic layout of all windows on the screen. Within WinManADL, the user assigns new attributes to each window using additional small squares located within its frame (figure 4).

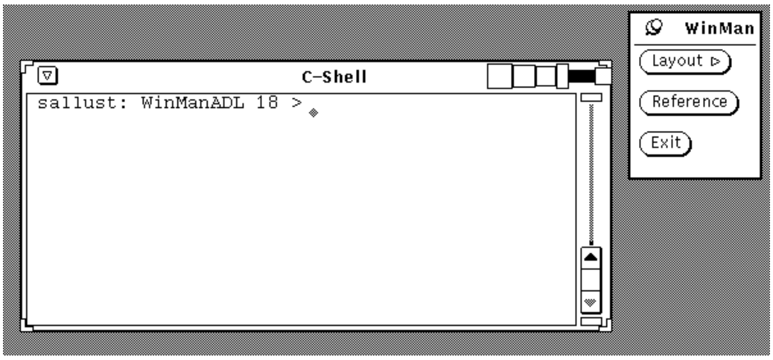

Figure 4. Additional small elements within the window frame for setting new layout attributes.

4. Conclusion

As a major step towards an improved interface of an information processing and multimedia system, we identified automatic display layout for graphical data representation and window placement. We implemented experimental tools to get experience with the requirements for display layout. Within these systems, we use general combinatorial optimization algorithms for layout computation. The benefit of using an algorithm independent cost function is the flexibility and tool independence of this approach. The flexibility is shown by the implementation of *GraphADL*, *HyperADL*, *SchematicADL* and *WinManADL*, each using his own type of cost function. Computing time data show that real-time execution is possible in next generation personal computers. Therefore, the introduction of display layout could contribute to another shifting of the limits of information handling within graphic user interfaces.

5. References

1. Christos H. Papadimitrou, Kenneth Steiglitz: Combinatorial Optimization, Algorithms and Complexity, Prentice-Hall, 1982.
2. Giuseppe Di Battista, Peter Eades, Roberto Tamassia und Ioannis G. Tollis: Algorithms for Drawing Graphs: an Annotated Bibliography, anonymous ftp by wlima.cs.brown.edu,1993.
3. Peter Lüders and Rolf Ernst: Verbesserung der Benutzeroberfläche von CAD-Systemen durch automatisches Bildschirmlayout, CAD 92, Springer, S. 77-93, 1992
4. 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, Germany, March 1993.
5. R.H.J.M. Otten and L.P.P.P. van Ginneken: The Annealing Algorithm, Kluwer Academic Press, 1989.
6. Thomas M.J. Fruchtermann and Edward M. Reingold: Graph Drawing by Force-directed Placement, Software - Practice and Experience, Vol. 21(11), Page 1129-1164, 1991.