A^{2}DL – An Adaptive Automatic Display Layout System

Stefan Stille, Shailey Minocha and Rolf Ernst

Institut für Datenverarbeitungsanlagen
Technische Universität Braunschweig
Hans–Sommer–Str. 66, D–38106 Braunschweig, Germany
email: stille@ida.ing.tu-bs.de

Abstract

With advances in computer technology, complex applications involve multi–window operations. With the increase in the number of windows on the desktop, effective viewing of all the necessary information for a task becomes difficult. A significant portion of the user's time is spent in opening/closing or moving/resizing the windows. Since these window interface housekeeping activities are not directly related to the user's task domain [1], time spent on window management causes an increase in the actual task completion time resulting in reduced user throughput.

We propose an Adaptive Automatic Display Layout System [1] *which would perform window management functions to free the user's cognitive and temporal resources to work on task domain activities rather than managing the multi–window desktop. Our system aims to automate and adapt the layout of the windows according to the current user and his current task domain, thereby enhancing the overall human–computer system performance.*

Keywords

Adaptive User Interfaces, Adaptive Window Management, Human Computer Interaction, Human Factors, Software Ergonomics.

1 Introduction

One of the consequences of the current development in the area of multimedia and hypermedia is that large amount of information has to be simultaneously displayed on the screen. User Interface (UI) software handles the output to the display screen and the input from the person using the computer system, who is called the *end–user* or just *user*, through windows. Windows enable the users to manage and access several sources of information associated with a single ongoing activity on one screen. Using the X System terminology [2], the UI software consists of two major window components: the window system and the window manager. The window system supports the separation of the screen into several windows while the window manager allows the user to move windows around, resize them, iconify or deiconify them, and is also responsible for displaying the title lines and borders around the windows. A window manager, thus, allows the user to organize the presentation of information on the screen to meet his needs. However, as the complexity of the applications grows, the user spends a significant part of his working time in arranging and resizing the windows on the screen to suit his current task or information requirement rather than working with the contents of the windows. A study performed by Bury et. al. [3] comparing users' performance in windowed and non–windowed systems suggested that the task completion time in windowed system can be longer due to the time spent in window arrangement. Their analysis indicates that although multiple windows reduce the user's short term memory load, and thus freeing more cognitive resources to the user's task but the benefits of windowing can be overshadowed by the extra time spent on window housekeeping activities.

In our previous work [4] at the Institute of Computer Engineering, Technical University, Braunschweig, Germany, we have proposed an approach of Automatic Display Layout (ADL), wherein certain layout algorithms could be used to automatically organize the windows on the screen to present the currently needed information in order to assist the user in his current activity. The motivation for automated window management by ADL is based on the premise that, by relieving the user from the burden of managing the window–based interface, the user throughput in his actual application domain activity is increased. This results in an improved human–computer system performance.

The problem of ADL has been formulated as a graph layout problem similar to the problem of chip layout in the area of VLSI [5] and has been treated as a combinatorial optimization problem [6]. We have proposed the usage of the simulated annealing [7] and the p–f (pressure – force) directed [8] algorithms to manage the display layout. The approach to ADL is not only applicable to windows or icons on the screen, but can also be applied to graphical objects within windows. This feature is especially important for hypertext or hypermedia applications. In order to demonstrate and test the suitability and effectiveness of these algorithms to the display layout problem, we have developed two experimental systems: GraphADL and HyperADL [9]. GraphADL is a graph browser that uses the proposed layout algorithms to enable the display of, and navigation in, a graph. HyperADL is a hypertext editor that computes layout for the hypertext cards that are to be displayed on the screen.

Fig. 1 is an illustration of the basic concept of ADL. The schematic layout in window 1 and the file–hierarchy

[1] This work is supported by German Research Foundation under grant no. Er 168/7-1

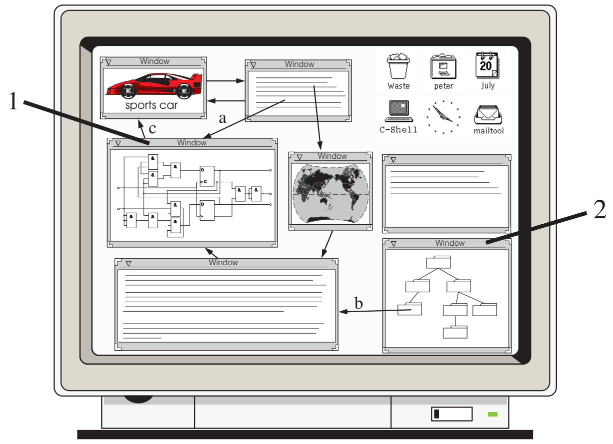

Figure 1: Complex Hypertext

layout in window 2 are two instances of the application of the proposed layout algorithms to graphical objects within windows. Window 2 is an example of a hierarchical directed graph layout. In addition, the windows on the screen are related (linked) to one another. Some of these relations are: A file in window 2 is displayed in the adjacent window through the link b. The schematic layout in window 1 is for a selected part of the car displayed in the first window and, as shown, is related to the textual displays in two other windows through the links a and c. All these relations (links) are considered when a layout computation is done to use the screen space efficiently and productively to suit the task.

The aim of our current work, that we discuss in this communication, is to apply the layout algorithms proposed in the ADL system to a real–time situation of human computer interaction. We propose an Adaptive Automatic Display Layout– (A^{2}DL–) System which would be able to adapt the layout characteristics, that is, placement and sizing of windows or icons, to suit the current user and his current working context. 'Current Working Context' implies the user's current task or working situation. The A^{2}DL–system learns during the process of user's interaction with the computer system and maintains a profile for each specific user which contains the learnt contexts and layout features for each context as per the user's preferred layout characteristics for that context. As we elaborate later in this communication, the layout features consist of the window/icon (object) attributes and the layout parameters, where the possible object attributes are size, width/length ratio, etc. and the layout parameters determine the characteristics of the display layout, like, homogeneous distribution of objects. The layout features are inputs to the layout algorithms for generating the layout. Thus, when the user is in a same or similar working context that has been learnt by the A^{2}DL–system and has been incorporated in the profile for that user, the layout algorithms would take into account the layout features for that context to present the layout. In future interactions with the user, the A^{2}DL–system not only learns new contexts and the associated layout features but also updates the information for the previously learnt contexts through feedback (layout corrections or modifications made by the user) from the user–computer system interaction.

As we discuss in Section 4, the learning process in the A^{2}DL–system is through neural networks as neural networks are useful for real–time pattern recognition. We have used the Adaptive Resonance Theory– (ART–) neural networks which are self organizing networks that have the ability to create a new pattern category when presented with a novel stimulus while preserving their previously learned knowledge [10].

To practically implement and demonstrate the proposed A^{2}DL–system, we are currently developing an Adaptive Window Manager (AWM) under UNIX/X11 over the public domain window manager fvwm [11]. One of our main concerns has been that the A^{2}DL–system should satisfy software ergonomic requirements, based on human factors perspective. The system should facilitate the ease of handling of the UI, increase user productivity, decrease user mental and physical workloads and, thereby, increase user satisfaction. Also, the layout computations should not be time–intensive as the purpose of the A^{2}DL–system is to support rather than interfere, to enhance user throughput and not to hamper the user's application domain tasks.

The paper is organized as follows: In Section 2, we discuss the layout algorithms as proposed and successfully demonstrated in the ADL system and the limitations of the ADL system. The software ergonomic requirements, as applicable to the A^{2}DL–system, and how we intend to meet them are elaborated in Section 3. In Section 4, we describe the architecture of the AWM and we present our concluding remarks in Section 5.

2 ADL System and its Extension to the A^{2}DL–System

As mentioned in the introductory section, the problem of display layout has been treated as the problem of placing the elements of a hierarchical graph, given by a set of nodes and a set of edges on a given placement area according to certain constraints [12]. The nodes correspond to the GUI objects and / or the graphical objects within the windows and the edges correspond to the semantic dependencies or links between the objects. The nodes have size attributes such as width, length, area and width–to–length ratio, and coordinates which determine their position on the placement area. An example of semantic dependencies between the nodes is the relation between the windows of an editor, a debugger and a textual command window where the compilation commands are being given, when the user is performing a debugging task.

The user, for effective viewing of the windows to suit his current task, may specify his preferences which may be location or position and size constraints, and his 'wishes' for the layout characteristics. The location constraints could be such as placing a window or a set of windows at some well defined positions on the screen. For example, the user may like a window to appear always at the top of another window or a window always in the top left corner of the screen. Then, a user may specify size constraints, like, he may specify fixed sizes for the window(s) on the screen. A user may also express his wishes, for example, desired sizes of the windows, minimization of edge lengths, that is, placing the related objects together on the screen, or homogeneous distribution of the graphical objects. Here, it may be emphasized that the constraints are minimum requirements that *have* to be satisfied (so far they are not conflicting amongst themselves) during the layout computations whereas the wishes should be considered during the layout calculations and may or may not be satisfied. Also, low computation time is an inherent requirement from a display layout system. Response time is a criterion for usability and quality of the adaptive display layout system as the process of layout should not delay the user's actual task domain activities. Apart from these inherent requirements and user defined preferences, we have formulated certain ergonomic principles for an adaptive layout system which we discuss in Section 3.

We have successfully applied combinatorial optimization algorithms for the display layout problem [4], [13]. In these algorithms, the computational result or the layout is controlled by the minimization of the cost function (also referred to as the 'objective function' in the literature). A cost function determines the quality of the displayed layout. It is expressed as a weighted linear combination of several components that determine the characteristics of the layout and represent the user's wishes for the layout. Each component of the cost function evaluates one attribute of the generated layout. Equation 1 describes the concept of a cost function: C is the cost function that needs to be minimized and is expressed as a weighted linear sum w_i of the cost of layout characteristics, L_i. The value of an individual component of the cost function indicates how far a user specified wish has been satisfied during the optimization of the overall cost function C.

$$C = \sum_i w_i L_i \qquad (1)$$

For the purposes of layout computations, a cost function has been represented as consisting of two components: static and dynamic. The static part contains functions which evaluate a single layout whereas the dynamic part contains functions that evaluate the changes of the current layout as to a given reference layout. The static part consists of the layout attributes of a single layout such as the edge lengths, edge crossings, overlapping of nodes and edges etc. The dynamic part of the cost function consists of two components: absolute changes between each node location in subsequent layouts, and differences in the relative placement of each pair of nodes in subsequent layouts. The dynamic part determines how closely the new layout matches the reference layout which is normally the preceding layout. The weighted linear sum of the static and dynamic parts yields a cost function. Since the display layout system has an inherent low response time requirement, it is important that the computation of the cost functions is not time–intensive. The complexity of the cost function determines the computation time and, therefore, we have considered simple cost functions that can be computed quickly.

The computation of a layout has been split up in two parts [4]: Global layout phase and the Postprocessing phase, in order to reduce the solution space for the optimization, thereby reducing the total computation time. During the global layout phase, simulated annealing algorithm, a well–known combinatorial optimization algorithm, is used to generate a coarse raster based layout to homogeneously place the objects in a grid–based way. The result of the global layout phase is a layout that is characterized by uniform object sizes and inefficient use of the placement area. The algorithm terminates when a precalculated estimated optimum of the cost function is achieved or when a given limit of the computation time is attained. In order to efficiently and productively use the display screen space, p–f directed algorithms are used in the post processing phase to iteratively change the position of the nodes to avoid overlappings, and the size of the nodes is changed by considering the desired size for each node. The force directed part of the p–f directed algorithm is responsible for minor changes in the position of the nodes while the pressure directed part introduces different node sizes. It may be noted that the basic layout characteristics, as evaluated by the global layout phase, are maintained in the post–processing phase with only minor changes in the positions of the nodes.

As there is a clear separation of the cost functions and optimization algorithms in the global layout phase, different algorithms can be used if they are able to meet the low computation time requirement. Thus, the system is flexible for the application of new and faster algorithms. We have explored the usage of genetic algorithm and applied it for the global layout phase [14].

Though the display layout problem has some special usability criteria regarding the low computation times and consideration of user's preferences, we have proved the applicability of combinatorial optimization algorithms by practically demonstrating them in the GraphADL and HyperADL systems. Experiments on these systems have shown that the computation times are within the range of seconds which will enable a real–time execution in the next generation work stations.

However, in the ADL system, the user is expected to specify many input parameters for the layout algorithms. The user has to specify the optimization criteria and hence the desired quality of the layout by indicating the weights of the static and dynamic parts of the components of the cost function. The user also specifies the object attributes, like, the desired size of the nodes for the p–f algorithms or the allowed overlapping of the nodes, or the upper bound of the computation time or the maximum number of iterations for the iterative refinement process of the layout to terminate. Thus, a user is expected to know about the working of the ADL system and about the layout algorithms.

In the A^{2}DL–system, in order to extend the usage of the algorithms of the ADL system to a real–time situation of human computer interaction, the emphasis has been to relieve the user from explicitly specifying any parameters for the layout algorithms. This is because a user is not expected to know the intricacies of the layout computations that take place in the background. Our aim is to develop an auto–adaptive layout system, a system that can learn about the user's preferences for the user's working contexts during the user's interaction with the computer system. The auto–adaptive layout system, without any direct interaction with the user, can also modify its learnt information when the user makes modifications to an already learnt layout for a particular context and thereby, present layouts for the learnt contexts which are appropriate for the user and his tasks. Thus, in the A^2DL–system, the user does not have to explicitly interact with the layout system to specify the desired layout characteristics for the layout algorithms but the system adapts itself to the user's needs and preferences through the learning process. Fig. 2 represents the A^2DL–system as a composition of the adaptation component and the ADL system.

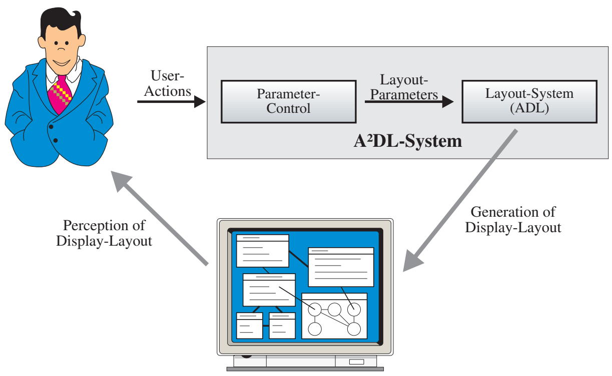

Figure 2: Concept of A^2DL

To achieve auto–adaptivity, in the A^2DL–system, we plan to consider the following while making the layout computations to generate a layout for any working context of the user: uniqueness of every window/icon in terms of its contents and the application from which it has been invoked and also the frequency of its usage. This is unlike the ADL system, where we treated each window/icon as a 'node' without considering its semantics. In the A^2DL–system, the uniqueness of each object is important for classifying and learning about the working context of the user, and for the process of layout adaptation whenever the user is in a similar or in the same working context.

In the next section, we discuss about the ergonomic requirements as applicable to an adaptive layout system.

3 Software Ergonomic Requirements

As we intend to apply the A^2DL–system to a real–time situation of user–computer system interaction and also because the display layout determines the UI's look and feel, it is important that the system meets the ergonomic requirements from human–factors perspective. The fundamental goal of human factors is to create a 'Usable System', that is, a system that optimizes user performance of tasks with an interactive computer system [15], increase user throughput, user satisfaction and user comfort. In order to construct a usable system, we have followed certain software ergonomic requirements during the design and implementation of the A^2DL–system.

In [16], the software ergonomic requirements for the design and evaluation of dialogue systems have been discussed. As no formal guidelines or norms have been laid down for adaptive systems, in general, and for adaptive layout systems, in particular, we have identified some of the ISO norms, DIN criteria formulated for dialogue systems [16] and certain guidelines in [17] and [18], and have interpreted them for adaptive layout systems. Our aim is to achieve these identified ergonomic principles in the development of the A^2DL–system.

We intend to apply the following software ergonomic requirements to our system: Availability, Comfort/Suitability for the task/Suitability for learning, Conformity with user expectations, Controllability, Suitability for individualization and Topological Consistency. Except for topological consistency, we use the same terms as expressed in [16], [17] and [18], but here, we describe the ergonomic requirements as applicable to adaptive layout systems.

Availability: Since the complexity of the layout algorithms to accomplish a particular layout is not known to the user, it is essential that the computation times for the layout algorithms are not too high. This requirement imposes a time limitation on our system for its acceptance and effective interactive usage. The main aim of our system is to save the user's time and enhance his performance in his current task by assisting him in managing the windows on the screen for effective viewing. Delayed response times of the system will cause stress and confusion to the user and thereby reduce his productivity.

Comfort, Suitability for the task and Suitability for learning: The A^2DL–system should support the user in the effective and efficient completion of the task by presenting a layout that matches his current activities. It is important that any corrections made by the user to the layout are correctly learnt by the system and reflected in future layout computations. The adaptive layout process should be task oriented in order to free the user from the activity of managing the user interface.

The interaction interface between the A^2DL–system and the user should not be complex as the user should not be expected to learn about the system like any other application on his desktop. In fact, the activities of the A^2DL–system should be carried out with minimal user involvement as the adaptive layout system is not a part of the user's task domain. However, the user should be knowledgeable to the extent that he can activate or deactivate the adaptation process as and when he desires.

Conformity with user expectations: In order to achieve conformity with user expectations, the learning process of the A^2DL–system should be consistent in accordance with the user's actions and preferences and the classification of the working contexts by the A^2DL–system should incorporate as precisely as possible the user's tasks. The adaptive system should then present similar and consistent layouts for all similar working situations of the user. Also, if the user has expressed any special constraints, they should be taken into account during the layout calculations.

Controllability: The user should be able to control the A^2DL–system at several levels: he should be able to activate or deactivate the adaptation process, he should be able to suppress the process of learning or he should be able to activate or deactivate the layout algorithms which generate layouts as per the learnt and/or adapted information. Also, the adaptation should be offered to the user in a form of a proposal that he can accept or reject and hence should be reversible. Thus, the user should be in full control of the system rather than the system controlling him.

The user should be able to specify any special position or size constraints or his 'wishes'. The 'wishes' expressed by the user should be converted into equivalent components of the cost function which is minimized during the process of optimization. We have practically demonstrated in the GraphADL and HyperADL systems that it is possible to take into account the user specified constraints and wishes as a part of the layout computations.

Suitability for individualization: In the A^2DL–system, the learnt profile should be user specific. The layout features, which are fed to the layout algorithms, should adapt to the user's individual requirements or preferences to a layout for a given task and thereby present display layouts to suit a particular user's needs for his task domains.

Topological Consistency: This is one of the most important requirements from an adaptive layout system. It is the measure of the preservation of a layout to a reference layout, which is actually the previous layout. The topology of the layout has to be similar in subsequent layouts so that the user is not confronted with sudden and inconsistent changes in the layouts in a new working context. Thus each new layout computation should take the previous layout as a reference and adjust the new layout as close as possible to this reference layout.

4 Architecture of the AWM

The architecture of the AWM is represented in Fig. 3. It is based on fvwm, a public domain window manager under UNIX/X11. AWM has been designed to function as an fvwm module. A module is an external program which runs as a separate UNIX process but transmits commands to fvwm to execute. The module does not affect the integrity of fvwm as its development, and communication with fvwm does not require any modification of the fvwm source code. Fvwm opens a pair of pipes for communication with a module [11]. One pipe is for messages to fvwm from the module, and the other is for messages to the module from fvwm. Each module has its own pair of pipes. After the pipes are open, fvwm forks and spawns the module. Fvwm can send messages regarding important window manipulations to the modules. This information contains, for each window, the window configuration, window, icon, and resource class names, and, if the window is iconified, the icon location and size. A module can also request fvwm to manipulate any window by sending messages to fvwm [11].

It may be noted that the layout capabilities of the system in Fig. 3 is limited to the layout of windows and icons on the display screen. For hierarchical display layouts within a window, individual applications will have to communicate with AWM. We are currently implementing appropriate communication facilities for our system. In this section, we will describe each component of the AWM and the

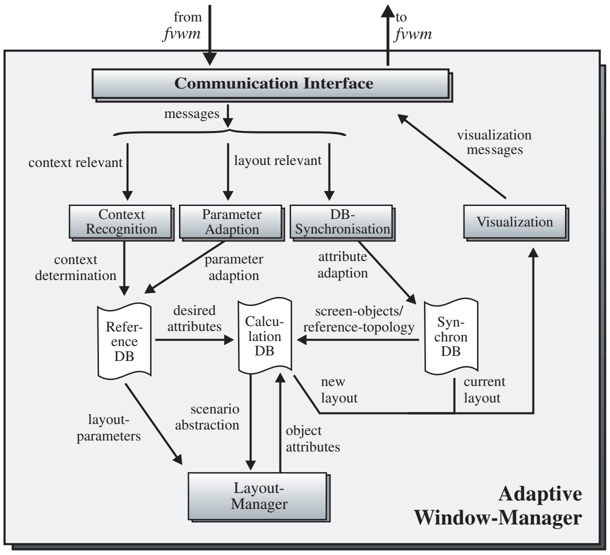

Figure 3: Architecture of the AWM

interaction between the components to yield the A^2DL–system.

AWM constitutes of the following components: Communications interface layer, three client components – Context recognition, Parameter adaptation and DB synchronization, a Visualization component, two databases – Reference database and Synchronized database which form the core of AWM, Calculation database which is a temporary database, and Layout Manager constituting the layout algorithms for layout calculations.

Communication Interface: This is an interface layer that separates fvwm from the other components of AWM in order to keep the underlying components independent of the window manager. It communicates with fvwm to send/receive the layout related messages to/from fvwm. The messages received from fvwm are in the form of message packets which have a detailed information content. For example, when a new window is created, 4 message packets are sent by fvwm which contain comprehensive information about the window name, resource name, resource class, etc. The communication interface summarizes the messages received from fvwm and puts them in a clear and a well–defined form so that the three client components can interpret the messages.

Context Recognition Client: This component contains an ART1 neural network [10], [19], that is responsible for classification and determination of the user's task domains or working contexts. It receives the context relevant messages, that is, messages related to the applications in a task that the user is involved in, from the communications interface layer, classifies them and assigns a unique context id, a positive integer, for each working context of the user.

DB–Synchronization Client: The function of this client is to receive the current layout relevant messages and modify the contents of the synchronized database so that the database is always in sync with the current layout situation on the display screen and always contains an abstraction of the current scenario.

Synchronized Database: The synchronized database contains the object attributes and the semantic dependencies between the objects of the current layout. The contents of the database are synchronized with the current layout on the display screen. The database contents serve as reference layout parameters for the layout calculations for topological consistency, when the layout algorithms compute a new layout so suit the changed working context of the user.

Reference Database: It contains the context ids and the corresponding user–preferred or desired layout parameters, like the weights of the cost functions for the global layout phase of the optimization, the forces for the force directed algorithms, etc., and object attributes, like, desired size of the window/icon, width/length ratio, area, etc. The contents of the database are user–specific and are inputs to the layout algorithms for generating a suitable layout for a particular context. At the start of the user–session, the database is populated by the context ids and related information from a file that contains the user–profile or the user knowledge base, that is, learnt context ids of the user's working contexts and their associated details. At the end of the user–session, the database contents are written to this file.

Layout Manager: The layout parameters and object attributes for the current working context are fed from the reference database to this component. The layout algorithms consider the current scenario stored in the synchronized database as a reference layout. The changes in the object attributes, as a result of the layout calculations, are maintained in a calculation database by the layout manager. The layout manager signals the end of layout calculations to the visualization component in order to bring about a change in the display layout as per the new object attributes to suit the current working context of the user.

Visualization Component: The visualization component gets the object attributes from the calculation database. It compares these with the object attributes stored in the synchronized database corresponding to the 'old' layout, which is actually the current layout and has to be modified. On the basis of this comparison, the visualization component generates layout commands in order to perform a step–by–step animation towards the new layout. These layout commands are given to a driver, one of its constituent, which sends the commands to fvwm across the communication interface to modify the position and size of the windows for the current working context. It may be noted that the visualization component interacts with the fvwm through a driver and does not generate fvwm messages directly as the internal components of the AWM are planned to be kept window–manager independent.

Parameter Adaptation Client: The user can make manual improvements to a layout, that had been generated by the layout algorithms, to suit his needs or due to changes in his preferences. These corrections may involve changes in the attributes of the screen objects, such as, changes in the sizes of the objects, iconification / deiconification of windows, or movement of the windows. The parameter adaptation client interprets the layout relevant messages received from the communications interface which indicate that a change in the layout has taken place due to the user's corrections to a layout and which is the basis for changing the layout parameters of the layout algorithms. On receiving such an indication, the parameter adaptation component picks up the object attributes from the synchronized database, which is always in sync with the current scenario, and maps them into corresponding changes in the layout parameters. The reference database is then updated for that context. Thus, whenever the user is in the same or in a similar working context, layout algorithms generate a user preferred layout. At present, we are investigating whether to use a neural controller/ regulator [20] or linear programming techniques for translating the changes in the layout made by the user into changes in the layout parameters that are updated in the reference database.

Working of the A^{2}DL–System

The context recognition module determines the context id for the current working situation of the user. There are two cases: a known context or an unknown context. A known context is that which the ART1 had learnt and classified during the user's previous interaction with the system and for which the data for the layout algorithms is already there in the reference database. An unknown context corresponds to a new working set of the user which the neural network had not learnt before and thus, a new entry in the reference database has to be created for the new context id. In either of the two cases, a context id is given to the reference database by the context recognition client.

If it is a known context, the layout parameters and object attributes for this context id are read from the database and fed to the layout algorithms in the layout manager. The layout algorithms get the current layout information from the synchronized database, which serves as the reference layout, and perform layout computations. These layout computations result in new object attributes which are maintained in a calculation database by the layout manager. After the layout computations, the layout manager invokes/signals the visualization component which interacts with the fvwm to generate a new layout to suit the new working context of the user (cf. Fig. 4).

When a new context id is received from the context recognition module, the reference database communicates with the synchronized database, which is in sync with the current layout situation, to obtain the object attributes belonging to that context. The layout parameters for the previous context which was prior to the new context of the user are retained for the new context as well in the database and are modified only when the user makes manual improvements to the layout for the new context. As in the case of a known context id, the layout features from the reference database are fed to the layout algorithms for layout calculations and finally, the visualization component directs

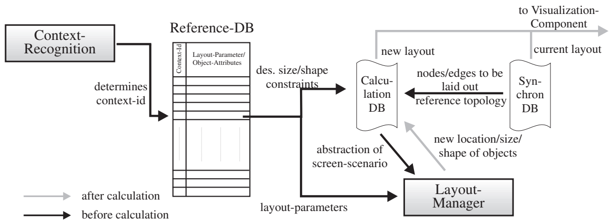

Figure 4: Data flow for layout computation

the fvwm to generate a fresh layout. After the parameter adaptation module learns from the user's corrections to the presented layout, the layout parameters of the reference database for that context are updated (cf. Fig. 5).

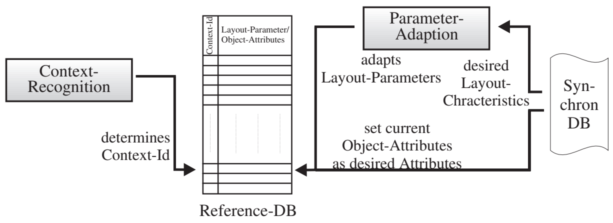

Figure 5: Adaptation of the reference Database

Thus the system is 'auto–adaptive' that adapts the layout features and performs layout computations to generate display layouts as per the user and his tasks without involving any direct interaction with the user. Also, the user being in full control of the system, has the possibility of suppressing the layout calculations by the layout manager or deactivating the learning or the adaptation process.

Our current work status and detailed analysis of each of these components will be communicated shortly in other communications.

5 Conclusions

Modern computer applications require multi–window operations. To simultaneously visualize all the required information for a task requires time–consuming window management operations by the user.

In this paper, we have addressed the problem of automating the task of managing the user interface in order to assist the user. We have proposed an automated adaptive display layout system which would organize the presentation of the windows to suit the user's current task. As the applications confronting users become more complex and information intensive, such an automated window management system offers great potential for enhancing user's performance by optimizing work loads, increase user comfort and satisfaction. The motivation has been to free the user from the burden of the task of managing the user interface so that he can apply his cognitive resources to his actual application domain activities.

We have discussed the motivation, software ergonomic requirements and the architecture of the proposed adaptive automatic display layout system. Detailed analysis of each of the modules in the architecture would be covered in our future communications.

References

[1] Ben Shneiderman. *Designing the user interface: strategies for effective human–computer interaction*. Addison–Wesley, 1992.

[2] Brad A. Myers. User interface software tools. *ACM Trans. on Computer–Human Interaction*, 2(1):64–103, March 1995.

[3] K. F. Bury, S. E. Davis, and M. J. Darnell. Window management: A review of issues and some results for user testing. Technical Report HFC–53, IBM Human Factors Center Report, San Jose, CA, June 1985.

[4] P. Lüders, R. Ernst, and S. Stille. An approach to automatic display layout using combinatorial optimization algorithms. *Software Practice and Experience*, 25(11), November 1995.

[5] Ernest S. Kuh and Tatsuo Ohtsuki. Recent advances in VLSI layout. *Proceedings of the IEEE*, 78(2):237–263, 1990.

[6] Christos H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, 1982.

[7] R.H.J.M. Otten and L.P.P.P. van Ginneken. *The Annealing Algorithm*. Kluwer Academic Press, 1989.

[8] Thomas M. J. Fruchtermann and Edward M. Reingold. Graph drawing by force–directed placement. *Software Practice and Experience*, 21(11):1129–1164, 1991.

[9] 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.

[10] 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.

[11] R. Nation. *fvwm – F (?) Virtual Window Manager for X11*. obtained by ftp-ftp://ftp.hpc.uh.edu/pub/fvwm/version-2/.

[12] P. Lüders and R. Ernst. The dynamic screen – beyond the limits of traditional graphical user interfaces. 13th IFIP World Computer Congress, Hamburg, Germany, September 1994.

[13] 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*, Los Alamitos, California, USA, October 1995. IEEE Symposium on Information Visualization, Atlanta Georgia, USA, IEEE Computer Society Press.

[14] P. Lüders and R. Ernst. Automatic display layout in window–oriented user interfaces. IFIP Workshop on Interfaces in Industrial Systems for Production and Engineering, Darmstadt, Germany, 1993.

[15] H. Reiterer. User interface evaluation and design. Technical Report GMD–Bericht Nr. 237, Gesellschaft für Mathematik und Datenverarbeitung, 1994.

[16] R. Oppermann, B. Murchner, H. Reiterer, and M. Koch. *Software – ergonomische Evaluation – Der Leitfaden EVADIS II*. Walter de Gruyter, Berlin, 1992.

[17] C. Baitsch, C. Katz, P. Spinas, and E. Ulich. *Computerunterstützte Büroarbeit, ein Leitfaden für Organisation und Gestaltung*. vdf–Verlag, 1989.

[18] P. Spinas. Interner Projektbericht des Lehrstuhls für Arbeits– und Organisationspsychologie (LAO). Technical report, ETH–Zürich, 1989.

[19] G. A. Carpenter and S. Grossberg. A massively parallel architecture for a self–organizing neural pattern recognition machine. *Computer Vision, Graphics and Image Processing*, 37:54–115, 1987.