Game Based Interfaces

Holger Diener and Hagen Schumacher

Fraunhofer Institute for Computer Graphics Rostock, Germany

Abstract

Complex menu and dialog structures prevent users from working intuitively with standard applications. In this paper, a new kind of user interface is suggested based on concepts of computer games. Defining a new structure for the functionality of an application will simplify users adjustment to new applications and will improve their everyday work. Usability tests based on eye tracking devices are described to verify the correctness of this game based interface approach. In addition, other game based concepts are introduced, which can be transferred to standard applications to improve the usability.

1. Introduction

Today, current applications occlude their functionality behind a highly complex menu and dialog structure. This seems to be necessary, because of the increasing scope of functionality that was integrated into applications during the last years. As a sales argument, it is important to offer as much functionality as possible. If experts still get along with these applications, then occasional users are deterred from such extensive applications. The vast number of possibilities suppresses any motivation to explore the application on their own. Reading the manuals before using the application is time-consuming and not much fun. Finally, users will switch to easier and simpler application, even if those do not offer all functionality that is needed.

It cannot be the intention of future applications to offer reduced possibilities for the one reason to keep the application clearly arranged and easy to use. Nevertheless, without new structures and concepts for user interfaces more and more functionality will shortly lead to unusable applications.

If we consider the entire software industry, a source for solutions can be found in the computer games section. In computer games, the rise of functionality is as high as in standard applications. For example, compare one of the first computer games *Pong* (1975) with a complex modern simulation games like *Ages of Empire*. In spite of complex functionality, computer games are still very popular. One reason for the popularity is certainly a different tenor towards games respectively standard applications. However, the most important reason is a better mediation of functionality using new concepts and ideas.

In this paper, we suggest a new approach for user interfaces in standard applications: game based interfaces. New concepts and ideas from computer games can reform the work with standard applications like word processing or CAD. The transfer of these concepts from game software or short *gameware* to standard software in everyday work or short *workware* will simplify users adjustment to a new application and will improve their everyday work. It is a good starting point to increase the efficiency of work.

Our first approach concentrates on restructuring the application functionality into new layers that depend on different situations or tasks. Each layer can be represented by different user interface structures. This approach is a result of observations of computer games. Some games use restricted user interfaces specialized on only a few tasks and change the interface whenever the situation or task changes. We think that restructuring the functionality respective the user interface of workware is a good way to improve everyday work. The restructuring methods will be described in more detail in the second section.

To compare game based interfaces with common user interfaces we will perform usability tests. More details about the tests and the test system will be given in section three.

In the fourth section, we will describe possible extensions and approaches for future work that deal with analysis of user behavior to generate personalized layers of functionality for every user.

2. Restructuring functionality

Many of today's application programs show that increasing functionality often contradicts good operability. We need a new structure, which offers the user only the needed functionality, to overcome the problems that arise from the common tree-like menu structure, like long access time for special functions, very long menus and confusing dialogs.

2.1. Basic concepts

This section gives an overview about the terms and objects mentioned in this paper. We will concentrate on the functionality of applications. Why do we need applications at first? Because, we have a number of problems, a work scenario, which have to be done, and we need a suitable application as a tool that supports us to finish the work. The application can fulfill this requirement if it contains the needed functionality. Therefore, we can differentiate between the *work scenario* as a basic motivation, the *functionality* we need to fulfill this scenario, and the *application* that provides the functionality. All three concepts are associated; if the scenario changes, we need new functionality and probably a new application. Because, today's applications are often oversized, the correlation between needed functionality and application is not one-to-one, we often use an application, which contains much more functionality than needed. How to provide simpler applications with only the necessary functionality, is the main concern of this paper.

We are not dealing with all problems of a scenario at the same time but one after the other. Our current problem is always a part of the whole scenario. It can be a scenario by itself, a sub scenario, or a single task. Therefore, we do not need the whole functionality necessary to solve the whole scenario, but only enough functionality to solve our current problem. Let us consider all functionality that we need to complete a work scenario as an abstract space. Sub scenarios or single tasks will then correlate to certain subspaces of functionality. These subspaces contain exactly the functionality that is needed for the given sub scenario or task. We call this subspace *functionally closed* to emphasize that also functions are included that are indirectly needed, i.e. needed by other functions.

Whenever our current problem changes we need other functionality to solve the problem. We can of course use an application, which is capable to handle all problems in our work scenario, i.e. which provides all the necessary functionality. However, this application would contain much functionality that is not necessary for the current problem and would be much more complex then a small application that is just suitable for the current problem. In smaller and simpler applications, it would be easier to find the functions and to perform the given tasks. In section 2.3 we will discuss two possible realizations to generate smaller applications which are suitable for the given sub scenarios: a building block principle and a filter method.

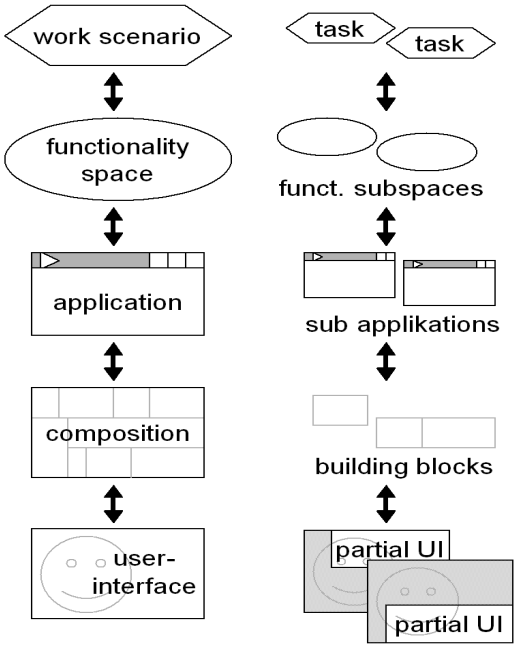

Figure 1: *Overview of the basic concepts*

2.2. Partial ordering of functionality

As mentioned above we need certain functionality to process all problems within a work scenario and all sub scenarios correlate to certain subspaces of this functionality. Let f_{max} be the functionality space, which is needed to complete the whole work scenario and let F be the set of all functionally closed subspaces $f \subseteq f_{max}$. These subspaces can be disjoint or have a nonempty intersection. Using the include relation, we can define an partial order $\leq$ on F with $f_1 \leq f_2$:if and only if f_2 contains all functionality of f_1, for all $f_1, f_2 \in F$.

f_{max} is the greatest element of this partial order and because only finite functionality is necessary to complete a work scenario, f_{max} is finite. Therefore, it is easy to see that for every two subspaces $f_1, f_2 \in F$ there is a smallest element $f_{sup} \in F$ with $f_1 \leq f_{sup}$ and $f_2 \leq f_{sup}$. With this we can define a supremum operator $\vee$ in F. If we also include the empty subspace of f_0, which theoretically represents an empty application, we can also define the infimum operator $\wedge$. With supremum and infimum operator the set F has all properties of a lattice.

2.2.1. Functionality nesting and suborders

For every partial order there is a bijective mapping onto a directed graph, therefore we can visualize the order by a Hasse diagram (see figure 2). The elements of the order are mapped onto nodes, and comparable elements of the order are connected by an edge, if there is no other element of the order between them. In the defined partial order, two elements f_1

and f_2 are comparable, if f_1 contains f_2 or vice versa. As the elements of the order are subspaces of functionality, the order defines a *functionality nesting* and every element is a nesting level. If every functionality space correlates to an application that provides this functionality, the order will define an application nesting as well. Again, if we can provide simpler applications with only the necessary functionality, we would get a direct correlation between tasks, functionality, and applications.

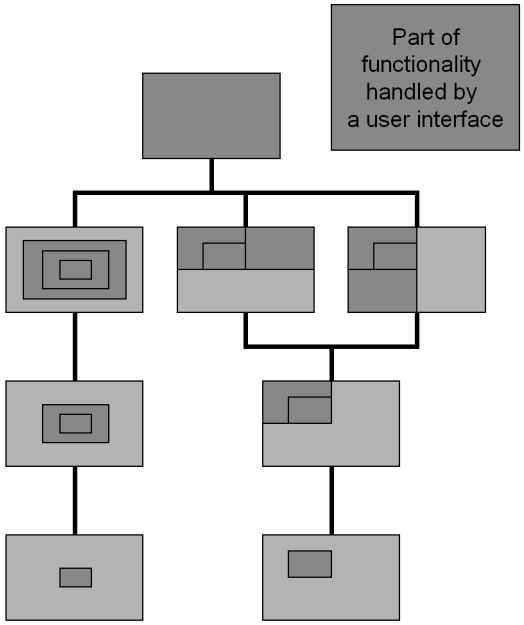

Figure 2: *An example for a partial order of functionality. The dark areas represent the functionality of the current level that is available by the user interface*

For every subset F_1 of F we can define a partial order by limiting the functionality order $\leq$ on F to the subset F_1. With this, *suborders* are defined within the functionality order. Each suborder can contain different functionality spaces and therefore it represents a set of tasks or sub scenarios. Two examples of suborders are given in section 2.2.2.

Some of the suborders are linearly ordered and can be represented by a chain within the graph structure of the functionality order. We can generate such a suborder by adding functionality step by step to a subspace with minimum functionality. The complexity will increase from level to level and will end with the greatest element f_{max}. Thereby each individual level element is a functionality subspace that represents a certain task or sub scenario and contains all subspaces of a lower level. If we consider applications correlated to the functionality spaces, this means, we will generate a hierarchy of applications that are suitable for the corresponding tasks. Of course, we can generate a linear suborder by starting with the entire functionality and reduce it from level to level until the desired or minimal level of functionality is reached.

2.2.2. Examples of suborders

Training software for a given application can serve as an example for a work scenario represented by a linear suborder or chain in the Hasse diagram. The entire functionality of a complex application is unsuitable for beginners. It would be a better way to start with a small functionality space and to switch to a higher level with more functionality whenever the user is used to the current one. In games, this is done on a regular base: Either you play level by level with increasing difficulty or you can determine a skill level at the beginning of the game. The game can be adapted to our skills. If we do not manage one level, we do not reach a more difficult level. This logical structure should also exist in learning programs. Only functionality should be offered, that is necessary and desired.

Thereby the principle of functionality nesting is very important to ensure that all functions already learned are available in higher levels. Rising in the suborder of functionality enables a user to learn the whole functionality of an application step by step. The presentations of the functionality spaces must also correspond with the functionality nesting, so that user interfaces on higher levels cover user interfaces on lower levels. Two realizations will be described in section 2.3.

Word processing software is another example for a functionality suborder. It can range from the functionality space of a simple text editor like "Notepad" to the functionality space of a complex application like "Word" with a multitude of functions. The functionality space of Notepad is the minimal element and the functionality space of Word is the maximal element of the suborder. To include more elements in the suborder, we can increase the functionality of the minimal element by adding more functions such as paragraph formatting, drawing elements or undo functionality. This will generate a functionality space at a medium level in the suborder, which corresponds to applications like "Write" or "Wordpad". Of course, there are many different suborders for the text-processing scenario. Depending on projects or tasks these suborders can contain more elements or can even be non-linear orders i.e. can have incomparable elements.

2.3. Game based interface

To increase the usability of workware we need simpler applications with only the necessary functionality to fulfill the current problem of the work scenario. This approach is a result of observations of computer games. Some games use restricted functionality specialized on only a few tasks and change the available functionality whenever the situation or task changes. Because we create a new kind of interface for

workware that originates in computer game concepts, we call this new approach *game based interface*.

The functionality order of section 2.2 can organize the existing functionality with respect to the given work scenario. This section will describe two possible technical realizations to create applications suitable for the different functionality spaces. First, we will describe the building block principle, which can be used to create a suitable structure when a new application is developed. Second, we will explain the filter method, which can be used to restructure existing applications.

2.3.1. Building block principle

The idea of the building block principle is the assembling of a new, suitable, and individual application from small independent software components: the building blocks. In this case, we are not restricted to the functionality of a given application. Some applications are using this approach by means of plug in software to extend their functionality. However, concepts to restructure the complete application or to adjust the user interface to a certain problem with small building blocks are still missing. A visionary solution would be the development of a highly component based system, which can integrate plenty of modules from different software producers. Users can purchase a basis system to provide a minimal set of functionality and can buy necessary extensions via the Internet later. However, this vision needs an extended cooperation of industry and research institutes to be fulfilled.

To test a simple variant of the building block principle we are developing a new desktop based on games to organize start menu and bookmarks. All programs on the computer together with all bookmarks will be the building blocks to generate an individual structure of the given data. An analysis of usage frequency with respect to the current problems can structure programs and bookmarks into different clusters. These clusters contain necessary applications and bookmarks to solve a special problem. For example, one cluster contains all tools and data to prepare a lecture. This includes presentation software, drawing utilities, and bookmarks for content retrieval. Another cluster contains tools to write a proposal. This cluster includes word processing software, calculation tools, and a bibliography. There can also be clusters for communication, time management, and many others. Therefore, most clusters represent functionality spaces that do not include each other, but are non-comparable elements of the order.

To organize the clusters, we will generate a city-map-like structure, with quarters specialized for certain problems. A small overview map allows fast and easy navigation within the structure to select each cluster. Because each cluster contains only a few programs or bookmarks, we get a fast access to all items. A visual authoring tool will be developed to generate the map and to arrange all items in this structure.

2.3.2. Filter method

The filter method can be used to restructure the functionality of an existing application on the presentation level. It creates a new user interface to control the application and to hide existing functions, which are not necessary for the current task or situation. Therefore, it represents a kind of filter that reduces the complexity of the application.

The implementation is done with an interface window that superimposes the original application and hides the old complicated menu structure. This interface window offers only controls for the necessary functions to solve certain parts of the given work scenario. Hereby, common controls are used together with new interaction methods like hotbox[2], rhythmic menus[4] and other concepts[3] [5]. We do not change the functionality of the given application directly, but we change the user interface to make functions available or unavailable. Therefore, each user interface correlates to a certain subspace of functionality. Changing the level of functionality can be done manually with special control elements in the interface window, which symbolize different problems or skill levels.

The working area of the application is still visible and can be accessed directly. The interaction with the working area is filtered to be able to react on users behavior according to the current functionality level.

3. Usability tests

We have developed a game based interface for workware in order to increase the efficiency of work, to simplify users adjustment to a new application, and to improve the relationship between user and application. We are carrying out different tests to verify that game based interfaces are providing these benefits.

First, we need a test to measure the performance of the new interface compared to a common one. This is done with usability tests based on the RealEYES system[1] developed in our institute. The RealEYES system makes use of an infrared camera to recognize the users point of view and compares it with the current mouse position. A complete screen capture and video records of the test person are made to monitor the whole test for a detailed examination afterwards. Based on an analysis of the relation of mouse position and point of view, the displayed content, and the video material it is possible to detect whether expectations of the users are fulfilled or whether the users are confused by the interface. With the RealEYES system, we are testing the same application with a standard interface and with a new game based interface. Thereby applications will vary from word processing software to research applications developed in our institute. A list of tasks will be prepared and will be given to test participants by a moderator. These tasks will vary in their difficulty and working time.

Second, the data we get from the RealEYES system must

be normalized with respect to the pre knowledge of the users. Therefore, we are also developing a questionnaire to collect more qualitative data additional to the quantitative data recorded by the RealEYES system. The questionnaire will ask about the pre knowledge of the user and the feelings concerning the program while accomplishing the tasks. By that, we hope to identify the influence of being already accustomed to a user interface. This factor is of great importance when we compare the quantitative results.

Additionally we would like to confirm a higher *fun factor* for the user while using a game based interface. This means that the relationship between user and workware improves. The questions about the users feelings concerning the program will provide the necessary data.

Comparing the qualitative and quantitative results for different kind of applications and different structures of functionality will generate reliable data to confirm the described benefits. These data will be available in autumn this year.

4. Future work

In the previous sections, we described an approach for a new structure of functionality. This approach bases on the level system of games and it is not the only one to establish gameware concepts in workware. Other concepts that can be used are the analysis of user behavior with corresponding reaction, story and screenplay concepts and personalized software. We will give some ideas on each of these game concepts and how they can be used in a working environment.

4.1. Analysis of user behavior

A linear suborder for training software as described in section 2.2.2 can also be used to develop a rating system for workware users. If the functionality of an application is nested, see section 2.2.1, we can evaluate the improvement of the user qualification very easily by looking at the current functionality level. A rating system could be more useful, if user behavior is analyzed automatically when working with the application. This analysis could also be applied to adjust an application to the current situation or task, instead of manual controls. In real time strategy games, the game engine analyzes the players behavior and reacts to it in an appropriate way. If we can transfer this concept to workware, the applications could guess the strategic goal of a user action and could respond for example with a more suitable interface.

4.2. Story and dramaturgy

One of the most powerful concepts of gameware is the story behind the game. Using a story makes it considerably easier to mediate the usage of an interface. If a button with a sword can control a knight in a computer game, we can certainly expect that the knight will fight when the button is pushed. It is much harder to mediate the meaning of buttons in workware. For some situations it is possible to invent a story for standard applications, again learning software is an example. In addition, the use of stories for certain functions makes it easier to remember the function. A slightly different method is the dramaturgy. Here it is possible to place some special effects within the user interface to change the users behavior in the right way. Like a film, the application manipulates the user actions.

4.3. Personalized software

A story establishes an emotional relation between player and game. The player immerses in a personal virtual world. We can also establish an emotional relation between a user and an application by changing the application in something more personal. This could be an individual design of buttons and controls or a new structure of functionality suitable for a special user. The personalized software helps to memorize the usage of the application and makes it more fun to work with. If we combine this approach with the individual analysis of user behavior, an application would be able to adapt itself to the user and would not expect the user to adapt to the application.

5. Conclusions

Within this paper, we have introduced the new concept of game based interfaces, which is a way to transfer concepts of computer games to applications of everyday work. We have described a method to restructure the functionality of an application in order to generate a restricted user interface, which is specialized on only a few tasks and situations and can be changed whenever the task or situation changes, see section 2.3.2. First observations have shown that using such interfaces can solve problems that arise from the common tree-like menu structures, like very long menus and confusing dialogs and therefore long access time to special functions. The usability tests will verify these observations and support further analysis of computer games.

An important property for the new structure of functionality is the functionality nesting, and its corresponding representation of nested user interfaces. This allows the changing of user interfaces when the level of functionality changes, because all control elements representing the functions of the application remain the same in place, function and design throughout all levels in which they occur. Therefore, the controls can be identified in every functionality level without difficulty. The graph representation of the partial order of functionality makes it easy to visualize different possibilities of user interface hierarchies and delivers a basis for a visual authoring tool for restructuring applications.

We have discussed two approaches for a technical realization of the game based interface: the building block principle und the filter method. Each of them can manage the defined

structure of user interface hierarchies. Thereby the building block principle is more suitable to develop new applications from small software modules and the filter method can restructure an existing application without changing the application itself.

Overall the concept of game based interfaces is a very promising approach to improve the usability of applications and much more concepts can be found in computer games for further improvements.

Acknowledgements

The approaches described in this paper are current research work within the project *Play the Application*. The project is sponsored by the INI-GraphicsNet Foundation.

References

1. Oertel, K., Hein, O., Elsner, A., "The RealEYES Project -Usability Evaluation with Eye Tracking Data", *Interact 2001 Eighth IFIP TC.13 Conference on Human-Computer Interaction Tokyo, Japan July 9-13, 2001 Waseda University Conference Centre, Shinjuku* 4
2. Kurtenbach, G., Fitzmaurice G.W., Owen, R.N., Baudel, T., "The Hotbox: Efficient Access to a Large Number of Menu-items", *CHI'99 Proceedings*, pp. 231-237 (1999). 4
3. Holland, S., Oppenheim, D., "Direct Combination", *CHI'99 Proceedings*, pp. 262-269 (1999). 4
4. Maury, S., Athènes, S., Chatty, S., "Rhythmic menus: toward interaction based on rhythm", *CHI'99 Extended Abstracts*, pp. 254-255 (1999). 4
5. Björk, S., Redström, J., "An Alternative to Scrollbars on Small Screens", *CHI'99 Extended Abstracts*, pp. 316-317 (1999). 4