GEDRICS: THE NEXT GENERATION OF ICONS

Jörg Geißler
German National Research Center for Computer Science (GMD)
Integrated Publication and Information Systems Institute (IPSI)
Dolivostr. 15, D – 64293 Darmstadt, Germany
geissler@darmstadt.gmd.de

KEYWORDS: pen computing, icons, gestures, gesture recognition, gedrics, human-computer interaction, user-interface design

ABSTRACT: Using today's combination of standard point-and-click user-interface elements for pen-based applications is a decision that implies that the pen is nothing more than a derivative of the mouse. This assumption is not necessarily correct. In order to be able to design more adequate interaction styles for pens, this paper introduces a new kind of user interface element: the *gedric*. Gedrics are gesture-driven icons, a combination of icons, pull-down menus and gestures. They provide a very fast, easy-to-learn, and easy-to-use interaction style for future pen interfaces. This paper describes and discusses the concept and implementation of gedrics.

INTRODUCTION

Like command languages graphical user interfaces (GUIs) have had to accomodate the increasing complexity of software functionality. When Xerox introduced the STAR system (Smith et al., 1982), direct manipulation, icons, and property/option sheets led the way to economic success and popularity through simple, more intuitive interfaces. However, these interfaces were simple for only a short period. It is very easy for software designers to invent new features and build them into the GUI. Thus, modern interfaces offer palettes for mode change, pull-down menus for hundreds of commands, context sensitive and hierarchical pull-down menus with dozens of items and much more. Today's complex software applications confront users with complex interfaces.

One reason for this increasing complexity is easy to identify: it is the point-and-click method. Most operations offered by the application are mapped on exactly one element in the GUI, e.g. on one icon, or one menu item. Users have to point and click with the mouse—for more than ten years *the* point-and-click device—more or less often to activate one of those operations (Foley & van Dam, 1984). For a small set of operations this method works fine but when the applications become more and more complex, the effort to activate these operations increases dramatically.

During the last few years, other input methods have become the center of interest. Electronic pens (Greenstein & Arnaut, 1988) still belong to the most promising of them. The first commercially available GUIs like 'Windows™ for Pen Computing' were still based on traditional concepts. Users did not click with the mouse anymore but tapped with the pen. And they tapped twice to open a menu and to select one of its items. Obviously, it was ignored that the pen is primarily a device for drawing and not for tapping. But in most of these interfaces drawing was reduced to scribbling free-hand notes. Pen interfaces require another interaction style.

This is why the idea of gesture recognition (Kurtenbach & Hulteen, 1992) emerge from theories about handwriting recognition. More or less complex drawings with the pen may be interpreted as gestures triggering special actions of the application. This aspect is not as easy as it looks like, neither in terms of the algorithms for the recognition process nor in terms of the cognitive overload for users who need to remember all the gestures. Gestures are only efficient if they are intuitive but how many gestures fulfill this requirement?

Nevertheless, the approach proposed in this paper is based on gesture recognition techniques because of its potential to minimize effort in the use of system functionality. One goal is that the functionality of a pen-based application may remain complex while the interaction itself should become very simple. Commands should be based on fast pen actions only, like a tap or the drawing of a stroke. Obviously, this idea is similar to the unistroke concept (Goldberg & Richardson, 1993) that tries to speed up handwriting. Gedrics are designed to speed up interaction.

To achieve this goal, we combine three already known concepts in the area of human-computer interaction to build a new kind of GUI element: the *gedric*, a gesture-driven icon. Gedrics are a combination of three components: icons, pull-down menus and gestures. We use icons because they do not need much display space, pull-down menus because they offer functionality on demand and gestures as short-cuts to reach this functionality.

The next section describes these three components in more detail. Then the concept behind gedrics will be presented, followed by some implementation issues. The paper concludes with a discussion, related work and future plans.

Geissler, J. Gedrics: the next generation of icons. *Proceedings of the 5th International Conference on Human–Computer Interaction (INTERACT'95)*, Lillehammer, Norway, June 27–29, 1995, pp. 73–78.

MOTIVATION

As with many innovations the starting point are problems with existing concepts. In our case it was the difficulty of equipping a meeting support tool (Streitz et al., 1994) with many operations that have to be accessed very fast and in natural way with electronic pens although the tool does not provide much display space for the GUI elements. We had a look at several standard methods but none of them would meet our needs (compare (Benbasat & Todd, 1993)). This section describes why.

Icons' Destiny

Most icons (Gittins, 1986) are nothing more than indicators for a specific process executed by an object in the application. Their images represent characteristics of these objects (associative icons) or they function as cognitive keys to them.

Like with any GUI element there are pros and cons for icons. Most of the literature stresses that their advantages outweigh the problems: they are easy to learn and to remember if there exist carefully designed images which are often embedded in a metaphor, and they save display space. Unfortunately, icons in applications, e.g. in palettes or tool bars, have one in common: they are constructed to be clicked on by devices like the mouse which activate their one and only operation. That is why complex icon-only applications need a large amount of them and why users have problems in remembering all their operations. Although it is not correct, the use of icons is often a synonym for having direct manipulation, but how poor is the idea of direct manipulation (Shneiderman, 1983) if users can only *click* on icons?

Pull-down Menus' Problem

Imagine the following scenario. A person working with a standard Macintosh™ text processor wants to emphasize a specific expression in a text by changing its font style to italic and increasing its font size. Thus, the person selects the passage, opens the font menu, selects the appropriate style item and does the same to change the font size. This seems to be little effort, but a more detailed look at the operations after the selection makes the hard work clear:

- First, the menu that holds the items the person is looking for has to be found. This is done by scanning the display and reading the menu titles.
- Then the mouse pointer is moved to the menu by dragging the mouse across the desk or mouse pad. The menu opens if the mouse button is pressed on the menu.
- While holding the mouse button pressed it is necessary to scan through the offered menu items until the one the person is looking for is found.
- Still holding the mouse button pressed the highlight bar has to be moved to the appropriate menu item.
- The highlighted menu item activates the corresponding operation when the mouse button is released.

Apart from the disrupting nature of this action sequence, the example shows how time consuming the scan-open-select method of menu-based systems is. All this work is required just for activating one operation. Therefore, it is not surprising that many of today's applications offer additional alternatives like keyboard shortcuts, or customized tool bars to access operations that are hidden somewhere in the menu tree.

Gestures' Profits for Pen Computing

Where the interaction style of the mouse ends with uncomfortable dragging procedures, the power of the pen just begins. To move the pen from one point of the display to another is one of the most natural actions one can perform with this device. Basically, all possible actions in a gesture-based user interface can be classified as follows:

Taps. For a long time point-and-click was essential for direct manipulation. In contrast to users who move their mice across a surface while their eyes are following the moving mouse pointer on the display, pen users point directly to the displayed object and tap on it. As long as there is no ''real'' access to the displayed object pen computing is pure direct manipulation even if the pen is used for tapping only.

Strokes. Drawing an almost straight short line between two points on the display is nearly as easy as tapping. Although this action seems to be very simple there is much more information hidden in a stroke. In addition to the location of the stroke on the display every stroke has at least two other characteristics: an orientation and a direction. Each of them makes it possible to differentiate between many simple and—from the result—very similar strokes, e.g. horizontal strokes drawn from left to right or vice versa, vertical ones from top to bottom or vice versa, diagonal strokes, etc.

Complex Gestures. Every drawing that can be interpreted by the application and that is more difficult to describe than a simple stroke may be seen as a complex gesture. As mentioned in the introduction, gestures are only powerful if they are intuitive. ''Intuitive'' gestures means that they are either already known to users (Raskin, 1994) because they reflect actions they did hundreds of times before with ordinary pencils or ball-pens, or they are simple enough to be learned at once. Intuitive pen gestures are, for example, the circling of a text passage to highlight or select it or the crossing-out of a paragraph to delete it.

If there is a ''new'' input device like the pen, software designers have the chance to develop new interaction styles. Icon-only interfaces for pen-based applications must fail because each icon covers exactly one operation and the operations are difficult to remember if the amount of icons in the interface increases. pull-down menus are good for hiding functionality but there is too much effort to select their items again and again. Gesture-only interfaces fail because most of the gestures are not intuitive and difficult to remember. Now, the issues are: Why should icons in pen-based applications only interpret clicks/taps and not other actions like strokes or more complex gestures, as well? How can the GUI help users to remember gestures? Is it possible to retain the concept of hiding functionality like with menus but to optimize the access

Geissler, J. Gedrics: the next generation of icons. *Proceedings of the 5th International Conference on Human–Computer Interaction (INTERACT'95)*, Lillehammer, Norway, June 27–29, 1995, pp. 73–78.

strategies? In short, is there a way to combine the best of different worlds, i.e. icons, menus and gestures, to model a new GUI element that is best for pen computing? The following section describes our approach to do so.

GEDRICS

A gedric is a *ge*sture-*dr*iven *i*con. It is an icon in a pen-based application that has the ability to interpret more gestures than just a simple click/tap. Remember the given example of emphasizing a selected passage of text. In order to do that with menus, one had to open them several times which was done with large effort. Realizing this functionality with a gedric could result in a ''font gedric'' on which users can draw gestures to manipulate several font characteristics. According to the example, to set the style of the selection to italic it would be possible to draw a diagonal stroke from the lower left to the upper right area of the gedric image. An almost vertical stroke from the bottom to the top of the image area would then increase the font size of the selection. Drawing both strokes the other way round would reset those values. This kind of user input is not only done very fast and with less effort but compared with icon-only solutions, there is also an enormous amount of display space saved. And there is no need for a special ''make italic'' or ''increase font size'' gesture.

Such an extension of the icon concept includes that the gedric image—in contrast to an icon image—may have to represent a large variety of functional possibilities of the underlying object. That is why a gedric image has much in common with a menu title. The operation that has to be activated is not only encoded in the image itself but it is always the combination of the image and a gesture. As a consequence, the representation of all possible actions of a gedric requires much more careful design of the gedric image than it is the case with icons.

Classes of Gedrics

Currently, we distinguish between two classes of gedrics: worker gedrics and container gedrics. Like many icons, *worker gedrics* process data, e.g. when they are activated by users. Most of them examine the current system status, e.g. if there is some selection, and then behave sensitively to the context. The main task of *container gedrics* is to store (processed) data. Although most of the functionality of a gedric-based application is realized with worker gedrics, container gedrics are necessary for the storage of (provisional) results during the working process or they handle external resources like file systems or peripherals.

Characteristics

This section is about those aspects that differ gedrics from any other kind of GUI element: their ability to recognize gestures, to support novice users of gedric-based applications, to store data, and to have children.

Most of the figures in this section show diagrams that demonstrate dynamic processes: idealized pen movements and their semantics to gedrics. To ''freeze'' the dynamics in a diagram, we choose the following notation (compare fig. 1): each square in a diagram symbolizes the area of a gedric image whereas the small circle with the attached line or free-form drawing visualizes the pen movement (the circle indicates the final pen position). A thick vertical line in some of the diagrams separates the gedric area from the editing area of the application.

Gesture recognition. The gestures recognized by a gedric can be divided into basic gestures that are the same for all gedrics (fig. 1) and those that offer special functionality. Current basic gestures are

- the tap that just activates a gedric. The result of the activation depends on the gedric's task. One gedric may simply undo an operation, another may do some scaling, yet another may print a selected text passage and so on. Basically, activated gedrics behaves similar to traditional icons.
- the stroke into its display area and back, symbolizing an arrow into it, a kind of *open* or *go into* command. Although every gedric recognizes this gesture, the result depends on the gedric class. Container gedrics will present their content whereas worker gedrics will open a pull-down menu in which users may select one of its items. The role of these menus is described in more detail below. It is important to mention that the corresponding diagram in fig. 1 demonstrates only one example how to make the open gesture. In fact, users may draw it from any angle into the gedric's image area.
- the drawing of a question mark without a dot. The resulting pull-down window not only explains any of the possible gestures, but also functions as an extended menu. Using this built-in help feature, novice users only have to know this one gesture to learn how to work with a gedric-based application.

Obviously, any of these basic gestures may activate a different operation, depending on the gedric's task. At first glance this may seem confusing (Grudin, 1989), but the principle is the same as with icons. Each icon in an application only understands clicks. Nevertheless, users have no problem with the fact that they may save, delete, print or scroll with that same kind of action. As long as the design of the gedric's feedback is done carefully, the activated operations will be transparent. More information about gestures will follow in the section on examples of gedrics.

Pull-down menus. Whenever a worker gedric receives an open command the appearing pull-down menu explains everything one can do with the gedric. As long as users are not familiar with all the gestures, this feature is very useful in their learning phase. At first, novice users may need the full description of the functionality using the built-in help, they may then only need the menu items and at a certain point they use gestures to have direct access to the operations without opening the menu. So gedrics support both

activate gedric *open gedric* *display gedric's help menu*

Figure 1. Basic gedric gestures.

Geissler, J. Gedrics: the next generation of icons. *Proceedings of the 5th International Conference on Human–Computer Interaction (INTERACT'95)*, Lillehammer, Norway, June 27–29, 1995, pp. 73–78.

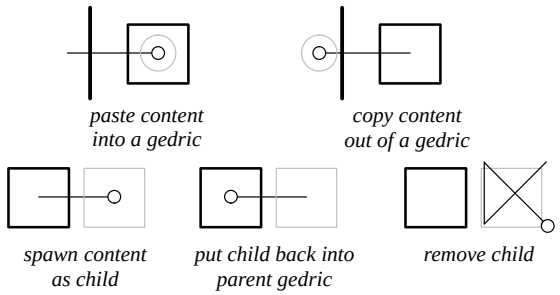

Figure 2. Gestures for container gedrics.

novice and expert users without any change of the input device as it is the case with keyboard shortcuts for mouse-based systems. They allow a smooth transition between different levels of user knowledge about how to access an application's functionality. Although the motivation for the design of gedrics was a pen-based system, they are suitable for mouse users, too, because they are also able to interpret mouse button events as *activate* or *open* commands. Thus, gedric-based applications can be used in environments of pen- and mouse-driven platforms.

Data Storage. Container gedrics behave differently than worker gedrics. Their repertoire of gestures is quite limited (fig. 2). Users can copy objects into a container gedric, e.g. the content of other gedrics, by pressing the pen on one gedric's image and dragging the appearing outline of that gedric's content to the image of the destination gedric. Whenever data of a container gedric is moved out of the gedric area into the application's editing area or vice versa, the application creates a copy of that data.

As an alternative to dragging, copy and paste operations can also be performed by tapping on (activating) a container gedric. It then looks for any selection in the editing area of the application and if it finds one, that selection is copied into its container. If there is no selection at all, the gedric produces an outline of its content that is then attached to the pen until users tap anywhere to drop the data. Although this method is not as intuitive as the first solution it is still an increase of interaction speed with nearly the same pen movements.

Parenthood. Another container gedric characteristics is the ability to spawn their content as an independent gedric. To do this, the data that is attached to the pen during a copy operation only has to be dropped somewhere else in the gedric area. After that, a new gedric will be created that also displays its content. That child gedric may be copied into the editing area of the application, moved within the gedric area, removed or even put back into its parent gedric. It depends on the specific task of the container gedric what will happen to its original content if one of its children is put back. Some merge the child's content with their own, others replace their content by that of the child. With container gedrics many system features can be implemented very easily. One example is a clipboard whose content may be set aside. Another is a marker gedric as it is explained in more detail in the following section.

CLIPBOARD (container gedric)
Stores data; ability to tear off content

MASTER OF TIME (worker gedric)
Multiple undo/redo functionality

DOORKEEPER (container gedric)
Import/export from/to file system and peripherals

PINBOARD (container gedric)
Marking of objects; ability to tear off content

TYPOGRAPHER (worker gedric)
Manipulation of font characteristics

ZOOMER (worker gedric)
Manipulation of object sizes

LAYOUTER (worker gedric)
Reorganization of display objects

Figure 3. Some design studies of gedric images.

Examples of Gedrics

This section provides an overview of gedrics that have already been designed by our group (fig. 3). Current gedrics cover functionality like loading and saving files, printing, scanning, editing, scrolling and more. Explanations of all of them would exceed the scope of this paper but the section provides exemplary descriptions of two gedrics to demonstrate the power of the gedric idea.

Layouter. A gedric that not only supports taps and simple strokes but also more complex gestures is the Layouter (fig. 4). Imagine an object-oriented drawing program. A person selects some of the graphical objects and wants to rearrange them. The Layouter now offers operations like snapping the objects to a grid, aligning them in several ways, and the person is even able to create a table-like layout by drawing a number on the gedric that stands for the number of columns that should be created.

Pinboard. Imagine a hypertext system (Nielsen, 1990) with many nodes containing text and graphics and linked together in a non-linear way, building a network. A person is browsing this net and finds interesting nodes for later

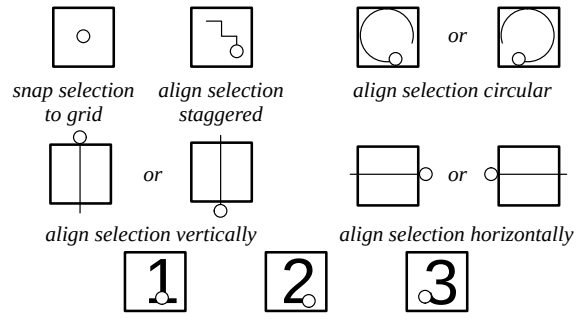

Figure 4. Layouter gestures.

Geissler, J. Gedrics: the next generation of icons. *Proceedings of the 5th International Conference on Human–Computer Interaction (INTERACT'95)*, Lillehammer, Norway, June 27–29, 1995, pp. 73–78.

visit. To mark those nodes one has to select them and tap on the Pinboard or drag them onto that gedric. The nodes are indicated by the application as marked and a formerly empty Pinboard shows its content. In this way it is possible to mark several locations in the net. Later, the person opens the Pinboard and it displays its contents in more detail. Each of the marked nodes appear as a gedric. The most important marker gedrics may then be dragged to a free place in the gedric area where they will appear as independent child gedrics. A tap on each of them triggers a jump to the corresponding node in the hypertext network. The marker gedrics may be removed from the gedric area very easily either by dragging them back to the Pinboard where they would be merged with the Pinboard contents or crossing them out which would delete them.

IMPLEMENTATION ISSUES

This section provides a general description of how gedrics can be implemented. Although they are also able to handle mouse events, the section concentrates on pen input.

Display Layers

Conceptually, we differentiate between two display layers: the working layer and the gedric layer (fig. 5).

- The *working layer* is responsible for the recording of the pen action, the display of temporary information, and the first step of gesture recognition. It behaves like a clear film on the display surface. People draw on it, their input is interpreted, maybe recognized or just kept as scribbles. Every interaction is based on this layer.
- The *gedric layer* is a ''deeper'' display level. The gedrics do not have any possibility to react directly to user input. They only get second-hand information through the transparent working layer.

Because of this distinction, it is possible to handle user input in a very flexible way. Gestures need not start or end exactly on the gedric image—the working layer can handle sloppiness. In addition, gestures are not restricted to be processed by exactly one gedric at a time—they can become relevant to more gedrics.

Gesture Recognition

The gesture recognition process takes place in two passes. The first pass checks the general characteristics of the input: whether it is an almost straight line with a specific direction and orientation, or a more complex graphical object. Then, this information is sent to the relevant gedrics which have to react according to their gesture understanding. The complete process is illustrated in fig. 5:

- Whenever the working layer detects a ''pen-down'' event it has to check whether the current pen position is on a gedric image or not. If it is, the result would be a mode change from ''recognize gesture'' to ''move gedric'' or ''copy and paste gedric's content.'' These procedures are much the same as those for traditional icon movement and dragging operations and will not be explained any further in this paper.
- If the pen is not located on a gedric image, the working layer begins to record the following pen movement by a periodical collection of pen coordinates. Furthermore, the recording includes the names of relevant gedrics, i.e. whose display areas are touched by the pen (those gedrics get highlighted for a moment). As long as the pen is still on the surface a line is drawn between those recorded points.
- A ''pen-up'' event stops the recording and starts the analyzing process. Apart from taps, the working layer is able to recognize strokes and a few other shapes. The recognized shape is then combined with additional data like significant points, e.g. the center of the shape's extent. This information is then sent to the next process which tries to find out which of the gedrics are relevant.
- For most shapes the center of their extent is the most significant point to analyze. If it is inside the display area of a gedric the complete shape information is sent to that gedric. If it is not clear which of the gedrics are relevant, the working layer does some clipping to the borders of the closest gedrics. The clipped shape with new significant points is then checked again. If there is still no possible gedric found the user input is rejected and a message is sent to the error handler.
- Finally, each gedric looks up in its internal dictionary whether it understands the received shape information or not and activates the corresponding operation. At this point one can call the user input a gesture with its task specific semantics. If there is no known gesture received, the application has to decide whether the gesture shape should be kept on the display as some kind of scribble or not.

To sum it up, the working layer only recognizes geometrical shapes and sends them to the relevant gedrics. Strictly speaking, a child gedric, for example, is not removed from the display because the user crossed it out but because the gedric understood a cross received from the working layer as a gesture to call the delete operation.

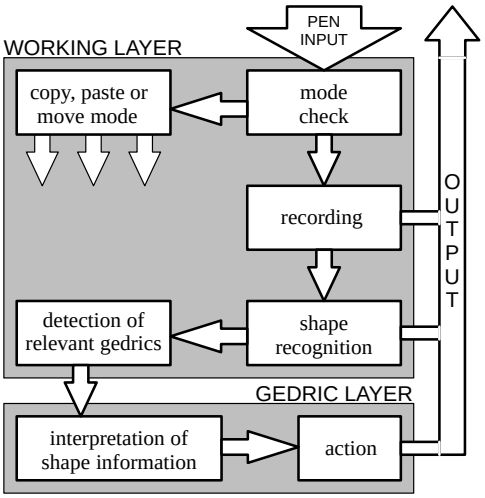

Figure 5. Gesture recognition process.

Geissler, J. Gedrics: the next generation of icons. *Proceedings of the 5th International Conference on Human–Computer Interaction (INTERACT'95)*, Lillehammer, Norway, June 27–29, 1995, pp. 73–78.

DISCUSSION AND FUTURE WORK

This paper introduced the concept of gedrics: icons that are primarily activated by pen gestures but that are also able to understand mouse events. Gedrics offer several different access strategies to system functionality for novice to expert users without any explicit mode change. They combine the advantages of icons, pull-down menus and gesture input.

The idea of extending the icon concept is not new. Attempts to add more information about the represented object have been published as *auditory icons* (Gaver, 1989) or as video thumbnails in a loop (*m*otion *icons*) (Brøndmo & Davenport, 1990). Using *earcons* (Blattner, Sumikawa, & Greenberg, 1989) the icon concept was even transferred to another medium. But none of these alternatives dealt with the aspect of how people interact with icons. These 'new' kinds of icons where still objects users point to (and click on). Nonetheless, these ideas will influence the evolution of gedrics. Future gedrics may be completely animated, offer sound as additional feedback, or even exist in another medium.

As mentioned before, even the design of traditional icon images may cause some problems and can impede the usability of an application. During our design studies of the first gedrics it was a problem to find an image that is able to represent a whole set of operations. Remembering real-world container-like objects helped us to find images for container gedrics but especially the design of worker gedric images is still a challenge. A guideline in this respect for us was to think of a representation of the object that has to be processed. The Typographer (fig. 3) that displays a plain character and that is able to manipulate font characteristics is an example that follows this guideline.

The majority of currently recognized gestures are stroke-based. The first reasons for this is that strokes as geometrical shapes are very easy and fast to process. Additionally, in combination with the gedric image strokes offer a large variety of possible operations that are accessible in a very fast way. More complex gestures not only require more pen movements but also more complex recognition algorithms that as a consequence usually require more processing time. Nevertheless, the repertoire of gestures each gedric is able to understand will be extended. We are also considering the support of multi-stroke gestures which would allow us, for example, to recognize complete question marks (with a dot) as a help gesture.

Gedrics are currently in the test phase. The first prototypes are implemented for the Newton™ and the first real application software will be a meeting support tool (Streitz et al., 1994) that is used in an environment where pen-based and mouse-driven platforms (electronic white-boards, personal digital assistants, notebooks, and personal computers) can be used side by side. Due to the limitation of display space on most pen-based systems and the fact that the standard meeting situation is familiar to a wide range of (non-)computer users, we believe that gedrics with their ability to support people with several levels of computer experience and their fast access methods to application functionality are the right choice for this kind of graphically-oriented application. The empirical results we will get from evaluating this software will then influence both the functionality of the application and the interface—the next generation of gedrics.

ACKNOWLEDGEMENTS

I would like to thank Jörg Haake, Gloria Mark, Ajit Bapat, Thomas Knopik, Chris Neuwirth and Norbert Streitz for their thoughtful reading of previous drafts and helpful comments.

REFERENCES

Benbasat, I. & Todd, P. (1993). An experimental investigation of interface design alternatives: icon vs. text and direct manipulation vs. menus. *Int. J. Man-Machine Studies*, 38, 369-402.

Blattner, M.M., Sumikawa, D.A., & Greenberg, R.M. (1989). Earcons and Icons: Their Structure and Common Design Principles. *Human-Computer Interaction*, Volume 4, 11-44.

Brøndmo, H.P. & Davenport, G. (1990). Creating and viewing the ElasticCharles - a hypermedia journal. McAleese, R.; Green, C. (ed.): *Hypertext: state of the art*. Oxford: intellect, 43-51.

Foley, J.D. & van Dam, A. (1984). *Fundamentals of interactive computer graphics*. Reading, MA: Addison-Wesley.

Gaver, W.W. (1989) The Sonic Finder: An Interface That Uses Auditory Icons. *Human-Computer Interaction*, Volume 4, 67-94.

Gittins, D. (1986). Icon-based human-computer interaction. *Int. J. Man-Machine Studies*, 24, 519-543.

Goldberg, D. & Richardson, C. (1993). Touch-typing With a Stylus. *InterCHI'93 Proceedings*, 80-87.

Greenstein, J.S. & Arnaut, L.Y. (1988). Input Devices. Helander, M. (ed.): *Handbook of Human-Computer Interaction*. North-Holland

Grudin, J. (1989). The Case Against User-Interface Consistency. *Communications of the ACM*. Vol.32 No.10, 1164-1173.

Kurtenbach, G. & Hulteen, E.A. (1992). Gestures in Human-Computer Communication. Laurel, B. (ed.): *The Art of Human-Computer Interface Design*. Reading, MA: Addison-Wesley, 309-317.

Nielsen, J. (1990). *Hypertext and Hypermedia*. San Diego: Academic Press.

Raskin, J. (1994). Intuitive equals familiar. *Communications of the ACM*, Vol.37 No.9, 17-18.

Smith, D.C. et al. (1982). Designing the Star User Interface. *Byte*, April 1982, 242-282.

Shneiderman, B. (1983). Direct Manipulation: A Step Beyond Programming Languages. *Computer*, August 1983, 57-69.

Streitz, N.A., Geißler, J., Haake, J.M. & Hol, J. (1994). DOLPHIN: Integrated Meeting Support across Liveboards, Local and Remote Desktop Environments. *Proceedings of CSCW'94*, 345-358.

Waterworth, J.A., Chignell, M.H. & Zhai, S.M. (1993). From icons to interface models: designing hypermedia from the bottom up. *Int. J. Man-Machine Studies*, 39, 453-472.

Geissler, J. Gedrics: the next generation of icons. *Proceedings of the 5th International Conference on Human–Computer Interaction (INTERACT'95)*, Lillehammer, Norway, June 27–29, 1995, pp. 73–78.