

User Interfaces of Interactive Statistical Graphics Software

Martin Theus
VIAG Interkom
Munich, Germany
martin.theus@viaginterkom.de

Abstract

Early graphical user interfaces were confronted with the attitude “*Real men don't use mice*”. In many cases this scepticism was easy to understand, since the design of early GUIs (graphical user interfaces) was even more cryptic than their text based counterparts.

Today, Apple and Microsoft have widely familiar interface design standards. They are similar but still different enough to be confusing. X Window System users are accustomed to being confused, since each application is likely to have its own idiosyncratic design style.

This paper will give an overview of some do's and don'ts in user interface design. Various examples will show that the implementer's intuition is often not enough and show the need of systematic approaches to efficient user interface design techniques.

Statistical graphics software benefits strongly from these design principles. Various interactive statistical graphics packages will be investigated to show how closely we can follow these principles in order to achieve a “good enough” design.

1 Introduction

Before looking at graphical user interfaces of statistical graphics software it is necessary to look at graphical user interfaces and design principles in general.

The paper will start with a brief review of several GUIs which were around in the last 15-20 years, followed with a *Hall of Shame* and *Hall of Fame* taken from the three major GUIs still in use today. With these examples at hand, we will then discuss the major design approaches. Before we look at four specific statistical graphics packages, we will list the most important do's and don'ts in user interface design.

2 History of GUIs

2.1 Early Mistakes

As with any new technology, it was not obvious at all in the beginning how things should be designed. Coming from text based systems and having only rudimentary graphical capabilities, the discrepancy between ideas and practical implementation was notably high. Figure 1 shows John Tukey in front of the PRIM-9 terminal. Certainly we cannot blame anybody for the



Figure 1: John Tukey in front of the PRIM-9 terminal

design resulting from these limited systems. But we should learn that almost any system still has some room for improvement. Figure 1 has been taken from ASA Graphics Video Library, which has its homepage at <http://www.research.att.com/~dfs/videolibrary>.

2.2 A History of User Interfaces

This section presents a brief overview and discussion of the graphical user interfaces of the last 20 years. Interfaces which are no longer in use are printed in *italics*.

- *Xerox Parc: The Star*
"The Star" — already unknown to most of the readers — was the first graphical desktop ever designed, and thus is the ancestor of any of the systems listed.
- *Smalltalk Desktop*
The Smalltalk desktop as well as the Oberon desktop only have been implemented together with their underlying operating system, which in turn was bundled to a specific programming language/environment. Thus, if one of the components is about to die, the whole system usually dies.
- *MacOS*
The Macintosh graphical user interface was the first one with extremely strict design rules. This resulted in a system, as Alan Kay put it, which was the first personal computer good enough to be criticized.
- *Atari ST*
The Atari ST was a bit like the uneducated little brother of the MacOS. Although having almost the same parents, the lack of design guidelines and reference applications prevented a broad success.
- *Special Workstation's Desktop*
In the late 80s almost any vendor of high performance graphical workstations came up with a variant of a graphical desktop — often based on a UNIX-like operating system. But all of these systems — like SUN's Sunview — had to surrender to X11.
- *X11*
Although X11 lacks a lot of the elegance and sleekness of its competitors, its openness made it the standard UNIX desktop.
- *Oberon Desktop*
As already mentioned, Niklaus Wirth did not learn from the mistakes of Smalltalk.
- *Windows (3.1, 95, 98, 2001, NT)*
As the Hall of Shame (cf. section 3) will show, there is no real reason why the design of the Windows desktop proved popular — but it did.

3 Hall of Shame

The Hall of Shame lists various examples of bad and misleading designs of components of graphical user interfaces. We will refer back to this section frequently in the section on design approaches.

3.1 Selecting the wrong control

Most elements of a graphical user interface are designed for a very specific purpose. So are check-boxes and radio-buttons. Check-boxes are designed to select an attribute or state of a component. Radio-buttons are used to select exactly one of mutually exclusive states.

The font attribute dialog inside *Word 6* (cf. figure 2) is a wild mixture of these two distinct controls, and the user gets no help in understanding which attributes may be mixed, and which not. Another good example of a

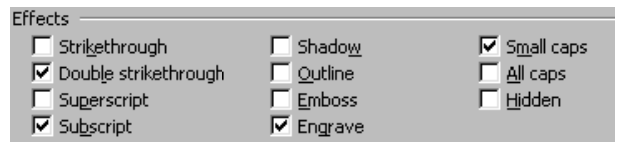


Figure 2: The font attribute selection box in Word 6

bad control are the so called "property sheets". Property sheets gather a whole lot of information — mostly options — grouped into different tabs. The problem with this interface is that the way how the tabs rearrange after clicking on a tab seems to be unpredictable for the user. Even if all programs chose the same way to re-

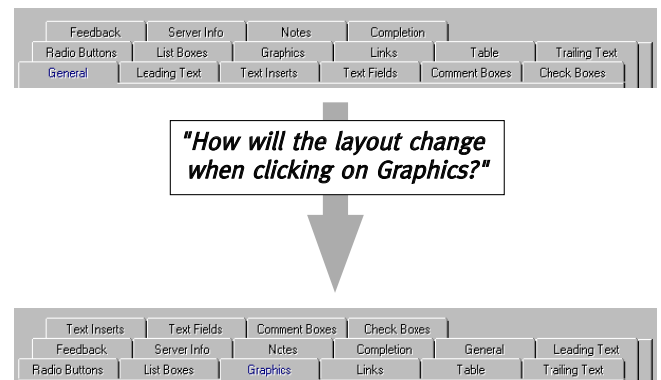


Figure 3: Property sheets confuse the user when trying to switch back and forth between two tabs.

arrange the tabs, the layout changes such that the user has to reread the titles of the tabs in order to find his way around. Figure 3 shows a particularly bad way to rearrange the tabs.

3.2 Misleading Error Messages

Ideally, good programming eliminates the need of error messages. When unavoidable, error messages should blame the system, not the user. They should point to possible causes and solutions. When giving advice they

should evaluate the error carefully to avoid messages like: *Can not delete file X. Please delete some files to free disk space.*

For example, when processing large amounts of data it is crucial to exactly point the user to where the error occurred. Imagine importing a 20MB data file, which results in an error stating that there is something wrong with the format, but not pointing to the exact position in the file.

3.3 Selecting the wrong Metaphor

Metaphors are a good means to transfer knowledge of real world processes onto the computer. The desktop is a good example for a metaphor. It is very important to carefully evaluate whether the metaphor suits the task on the computer or not. Computer scientists thought about extending the desktop metaphor to a whole city, which includes shopping malls post offices etc. as we are used in the real world. Whereas this is certainly a valid metaphor, the overhead of navigating through this imaginary city is far too big. Tasks like shopping or getting and sending mail can be mapped much easier to simpler elements of a graphical user interface. The Macintosh

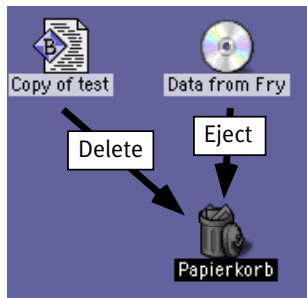


Figure 4: The trash bin metaphor on the macintosh is ambiguous.

uses the trash bin metaphor to delete files. Files can be moved into the bin, but are only gone after the bin has been emptied. Unfortunately, the designers of MacOS also used the trash bin to eject disks resp. unmount volumes; or to be more precise to remove them from the desktop. Volumes of course are not delete by dragging them into the trash bin.

3.4 Inconsistency

Inconsistency is maybe the worst problem in user interface design. From children we know that they like everything to be at its right place at the right time. When using computers we have the maturity of children. Using software gets much easier when similar tasks can be

performed with similar commands. If things look the same they should behave the same. Figure 5 shows the

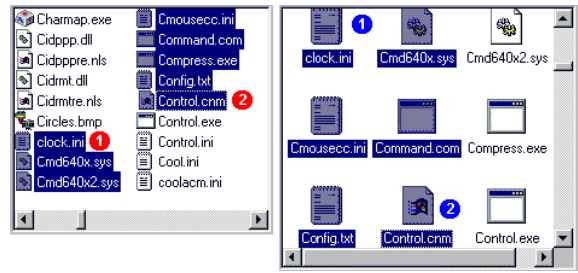


Figure 5: Basic selection mechanisms inside the Windows interface do not behave consistently.

selection of files within windows of a Windows desktop. The left window shows a selection in the list-view, the right windows a selection in the icon-view. Whereas in the list-view the items are selected columnwise within the first and last element selected, in the icon-view all items in the rectangle which is formed by the upper left and the lower right file are selected.

4 Hall of Fame

Unfortunately, the hall of fame is much smaller than the hall of shame. In recent years, only a few concepts were invented that proved popular. Maybe the best example is the most recently used (MRU) concept. Figure 6 shows an application which prompts all recently used files in a submenu. In doing so the software reduces the memory load from the user and moves it onto the computer — where it can be handled much easier.



Figure 6: The concept of most recently used items is very helpful.

Other improvements in graphical user interface design are tooltips, which make the navigation through icons and other widgets much easier, without the need of a manual.

Similar to tooltips are cues. Whenever the cursor mover over a hotspot, the shape of the cursor will change to another icon. The new icon gives a hint of a function which will be performed when clicking (and dragging) the mouse. The most common use of cues is maybe the

resizing of windows within X11 window managers, but also can be found in various other — often graphical — applications. Similar to cues, and probably older, is the simple notion that the cursor changes to reflect the function of a window or widget it is inside: The I-bar is uniformly used to represent text input, for instance.

Previews are very helpful when searching for a particular file. In the file selection box, the contents of the selected file will be displayed in an icon view (for graphics files) or the first few lines of a textfile are listed.

5 Design Approaches

5.1 Principles

These seven design principles summarize Norman's (1988) approaches which apply to everyday things.

1. **Use both knowledge in the world and knowledge in the head.**

A good design should incorporate the knowledge of the user based on his real world experiences as well as the knowledge based on abstract evaluation.

2. **Simplify the structure of tasks.**

Obviously any task should be performed as simple as possible. An example of a successful simplification of a task is drag & drop, which simplifies copy & paste by two steps.

3. **Make things visible.**

The design of a tool should make its functionality obvious, and visible feedback should tell the user about the state of the system.

4. **Get the mappings right.**

The layout and position of controls should be mapped as closely as possible to the objects being controlled.

5. **Exploit the power of constraints, both natural and artificial.**

Constraints can make things much easier. Offering only the options which are actually available will simplify an interface.

6. **Design for error.**

Although a design should avoid as many errors as possible, errors which can not be avoided should be handled as graceful as possible.

7. **When all else fails, standardize.**

Standardization might be the simplest principle, but also the most powerful. Standards contribute a lot towards flatter learning curves and will naturally help to avoid errors.

5.2 Heuristics

These heuristics — taken from Nielsen (1993) — are more specifically tailored towards graphical computer human user interfaces.

- **Visibility of system status**

The system should always keep users informed about what is going on, through appropriate feedback within reasonable time. Progress bars do not shorten the time the user waits for a task but they definitely make the user feel more comfortable.

- **Match between system and the real world**

The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

- **User control and freedom**

Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

- **Consistency and standards**

Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

- **Error prevention**

Even better than good error messages is a careful design which prevents a problem from occurring in the first place.

- **Recognition rather than recall**

Make objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Reduce the users' memory load.

- **Flexibility and efficiency of use**

Accelerators unseen by the novice user may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

- **Aesthetic and minimalist design**

Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

Do ...	Don't ...
... talk to the user	... believe your intuition
... present and test prototypes	... ignore the user's feedback
... iterate your design	... be lazy
... think in terms of tasks	... think in terms of the system
... maximize consistency	... be satisfied with a local solution
... think in reusable objects	... write code over and over again
... conform to prevailing standards unless there is a good reason not to	... reinvent the wheel
...	...

Table 1: Some important Do's and Don'ts in user interface design

- **Help users recognize, diagnose, and recover from errors**
Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

5.3 Aesthetics

Aesthetics is hardly measurable on an objective scale. Although thus being not tangible, aesthetics often results in the right “look and feel” of a user interface.

A consistent design usually will build a harmonic application. Using well defined components the program will build up more easily. Obviously object oriented programming is the key technology.

5.4 The Process

- **Early focus on users and tasks**
Study the attitudes and behaviors of users, and become familiar with the nature of their tasks. Many research software tools might not be targeted towards a specific user community. In these cases the developers — which are also the primary users — should involve potential other users in the design.
- **Empirical measurement**
Early in the development process, study the performance and reactions of the system's intended users using simulations and prototypes.
- **Study your GUI carefully**
Be sure you know all components of the graphical user interface and their correct use. The choice of the right component might not at all be obvious.

- **Talk to colleagues**
Always keep in mind that the developer's ideas are just a sample of size one. Exchange your ideas in order to find the most appropriate solution: “Maybe both of you are wrong!”

- **Iterative design**
Keep redesigning the system as long as user testing reveals problems. Try to find the right balance between ease of use and offered command complexity.

6 Do's and Don'ts

With the design approaches presented in the last sections, we can now formulate the most important do's and don'ts for the design of graphical user interfaces. The list (cf. table 1) is certainly not complete which is indicated by the horizontal dots in the last line.

7 Examples

This section discusses four examples of interactive statistical graphics software and highlights their strengths and weaknesses.

The applications chosen here are — apart from the fact that they are best known to the author — some of the most advanced tools for a graphical exploration of data.

Other statistical packages like S-Plus, SPSS and SAS offer graphical front ends as well which are just mere translations of the underlying text-based command line interface. The graphics offered in these packages are mostly static and do not allow for any interaction. Therefore these packages are not considered here.

7.1 DataDesk

DataDesk (Velleman, 1997) is the only commercial package in this section. Obviously the developer was very much influenced by the idea of the desktop concept. Every object (e.g. (relational) data, plots, statistical models etc.) can be arranged in folders on the desktop. Elements of an analysis must be removed by using the trash bin. The work which was done during a session in DataDesk is stored in the datafile and thus can be identically accessed in a later session. This is very important when using the desktop metaphor.

A drawback of DataDesk's design is the inconsistent handling of color assignments. Data represented by points (e.g. in scatterplots) reflects the color assignment correctly, whereas data represented by areas (e.g. in barcharts) does not. Another inconsistency can be found with the selection tools, which do not apply in all plots. Although brushing a piechart does not seem to make sense, brushing a histogram would.

DataDesk uses special forms to display statistical

models. These forms can be implemented with standard elements of MacOs as well. It is not clear why the developers did not conform to prevailing standards.

Windows inside DataDesk have an additional box to allow drag and drop windows directly and not just the icons of the window. This functionality proves to be helpful and is a valuable extension to standard graphical desktops.

Cues in DataDesk are called HyperView menus. These menus are often hidden, i.e. there is no visual clue where to find HyperView menus other than the sudden change of the cursor shape.

The graphical interface to set up statistical models in DataDesk is not quite intuitive. Although the drag and drop interface to add and remove variables from the model seems a good choice, a modeling interface like the one used in S-Plus seems to be more natural. The S-Plus interface uses a language which is very close to the formulas we learned in statistics class and thus is much closer to the way most people think of statistical models. A combination of both approaches maybe the best choice.

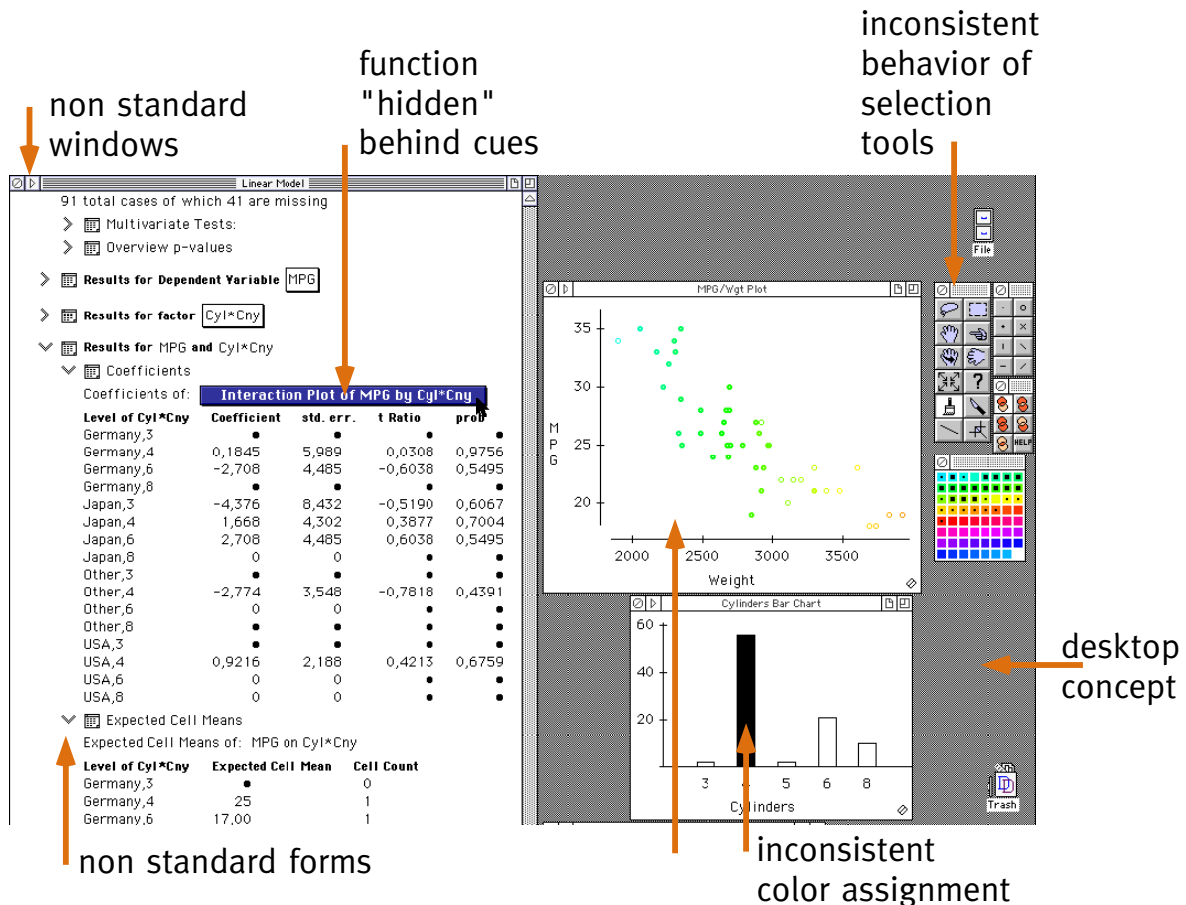


Figure 7: A sample DataDesk desktop

7.2 MANET

MANET (Unwin et al., 1996) is research software. Since research software usually focuses on new ideas, many indispensable features of commercial software are omitted there. Thus MANET does not offer any desktop or session concept. Instead MANET is tailored towards an optimized use of direct manipulation tools. The meaning of direct manipulation interfaces is often misunderstood. Direct manipulation does not usually mean being able to manipulate objects by specifying or changing their parameters, but direct interaction with the graphical object. For example, the number of bins in a histogram can certainly be reset by opening a parameter dialog and typing in the new number of bins. With a direct manipulation interface the user is able to select a handle on the histogram and drag the mouse, watching the histogram continually update itself as the number of bins is reset with every move of the mouse. MANET offers several of these direct manipulation interfaces. Categories in discrete variables can be reordered by drag and drop actions in a barchart, x- and y-axes can be flipped in

a scatterplot by a single mouse click, etc. A drawback of the implementation is the fact that these direct manipulation interfaces are based on cues, which — for the novice user — are often hard to locate. Furthermore the extensive use of cues is not part of the standard Macintosh user interface, and has only partly been introduced with MacOS 8.

Another aspect of direct manipulation interfaces are queries. Almost every object in MANET can be queried by an option-click. Queries can be extended beyond data which is actually plotted, and any variable in the dataset can be used to be displayed in a query (shift-option-click). An often neglected design principle is to give feedback to the user on the system status. MANET offers red warning flags in the plots, if data is not visible because of either being outside the plotting canvas or of limited screen resolution. Since interactive graphics is build on the paradigm of linked highlighting, there should be sufficient feedback for the user, which data was selected. Selection sequences (Theus et al. 1997) with the corresponding selection area offer feedback which is indispensable in complex hierarchical selections.

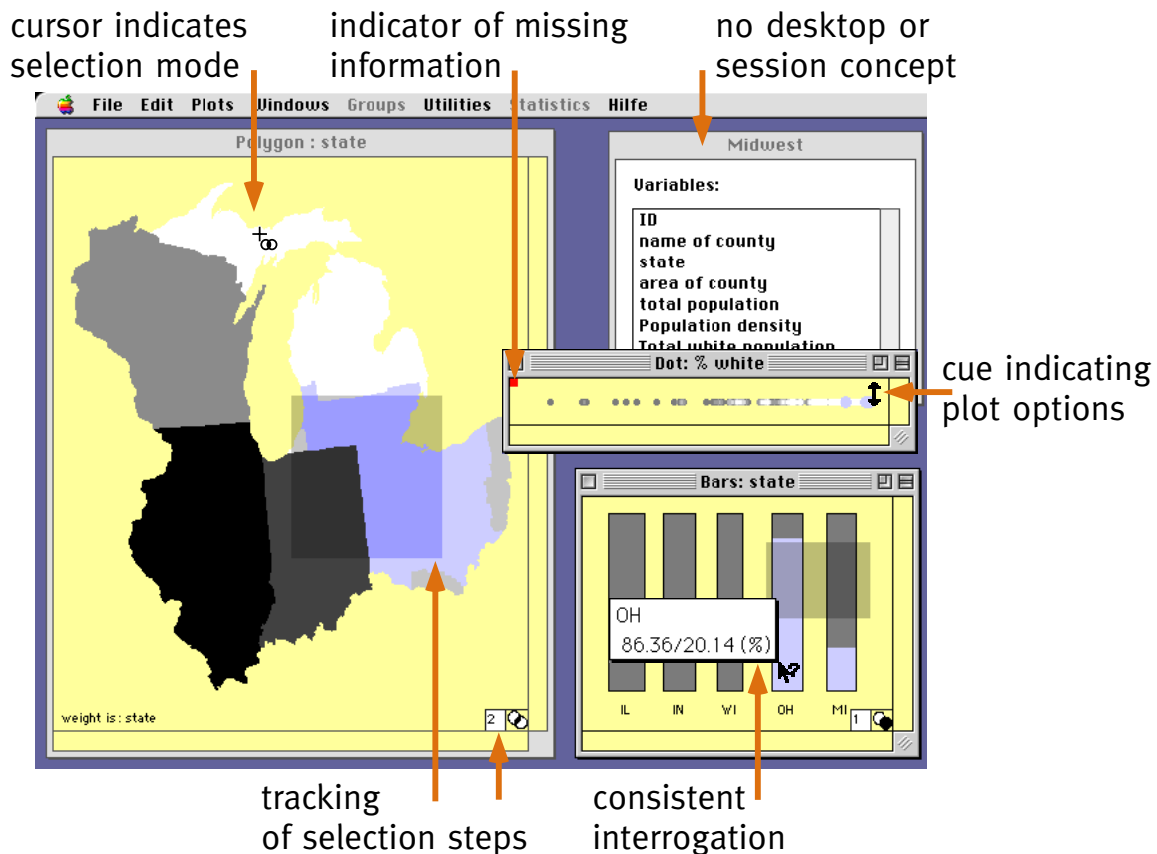


Figure 8: A sample MANET session

7.3 Mondrian

Mondrian is an application developed in pure JAVA (Theus, 1999). Thus it runs under any platform which supports JAVA. Since graphical user interfaces differ substantially between platforms, the JAVA Swing packages were used to build a uniform look and feel independent of the underlying platform.

Mondrian as well as MANET is research software and thus no attention has been drawn to the implementation of a desktop or session concept. In contrast to MANET the direct manipulation interface of Mondrian only uses elements of the Swing interface. Instead of cues Mondrian utilizes pop-up menus to implement direct manipulation operations which can not be performed on graphical objects (e.g. bars, tiles, points, axes) themselves.

Mondrian's main focus is on categorical data and advanced selection mechanisms. In order to give the user maximum feedback on what he or she actually selected,

each selection step is visualized by a rectangle. Via these selection rectangles the selected subset can be modified by either resizing the rectangles with one of the eight handles or by choosing a new selection mode (i.e. one of: replace, and, or, xor, not). Only the selection which was modified last is plotted in a black outline, the remaining are drawn in a lighter gray. This selection concept has proven very useful when working with complex selections but is still flexible enough for single or two step selections.

A very simple but yet powerful example of a combination of graphical and textual feedback is the selected subset display at the bottom of the variable window.

In order to flatten the learning curve as much as possible all actions have been consistently mapped to mouse/modifier combinations. This is achieved by including this functionality into the basic JAVA classes on which all plots are built. Therefore all plots inherit these controls and consistency is guaranteed.

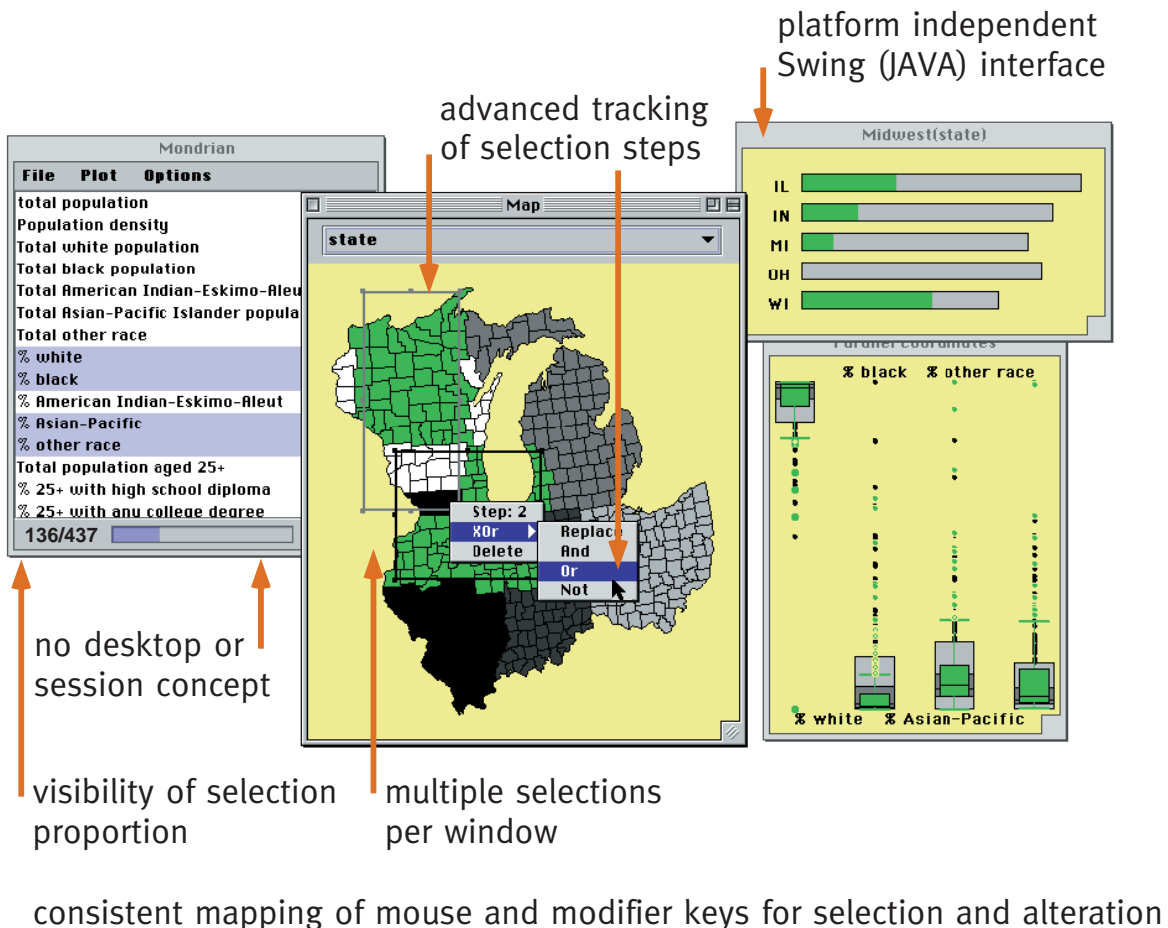


Figure 9: Analyzing the Midwest data set using Mondrian

7.4 XGobi

XGobi (Buja et al., 1996) — in contrast to MANET and Mondrian — was designed to investigate high dimensional *continuous* data. Thus we need controls to select variables quickly and provide visual feedback of the current projections of the selected variables. This is implemented by variable circles in the right panel. Whereas the visual feedback is excellent the selection of variables is hard to learn and sometimes confusing.

The left panel in XGobi is a good example of natural constraints in user interfaces. It only offers options which are actually available in the currently selected operation mode. Although this reduces the number of possible options dramatically this panel is still cluttered in some modes.

Unfortunately, XGobi uses an old and non-standard X11 interface toolkit; many conventions that are now familiar do not apply here. For example, the common hints to distinguish push buttons from radio buttons from “choice buttons” do not exist in the Athena widget set used to build XGobi. Another unfamiliar and thus confusing control are windows that solicit text input in a non-standard fashion.

Maybe the worst drawback of XGobi is the fact that in order to get a linked view of multiple plots one has to restart (clone) XGobi with the same dataset several times. Each of the windows again offers all controls which fills the computer screen very fast, and thus makes working with multiple views a cumbersome task.

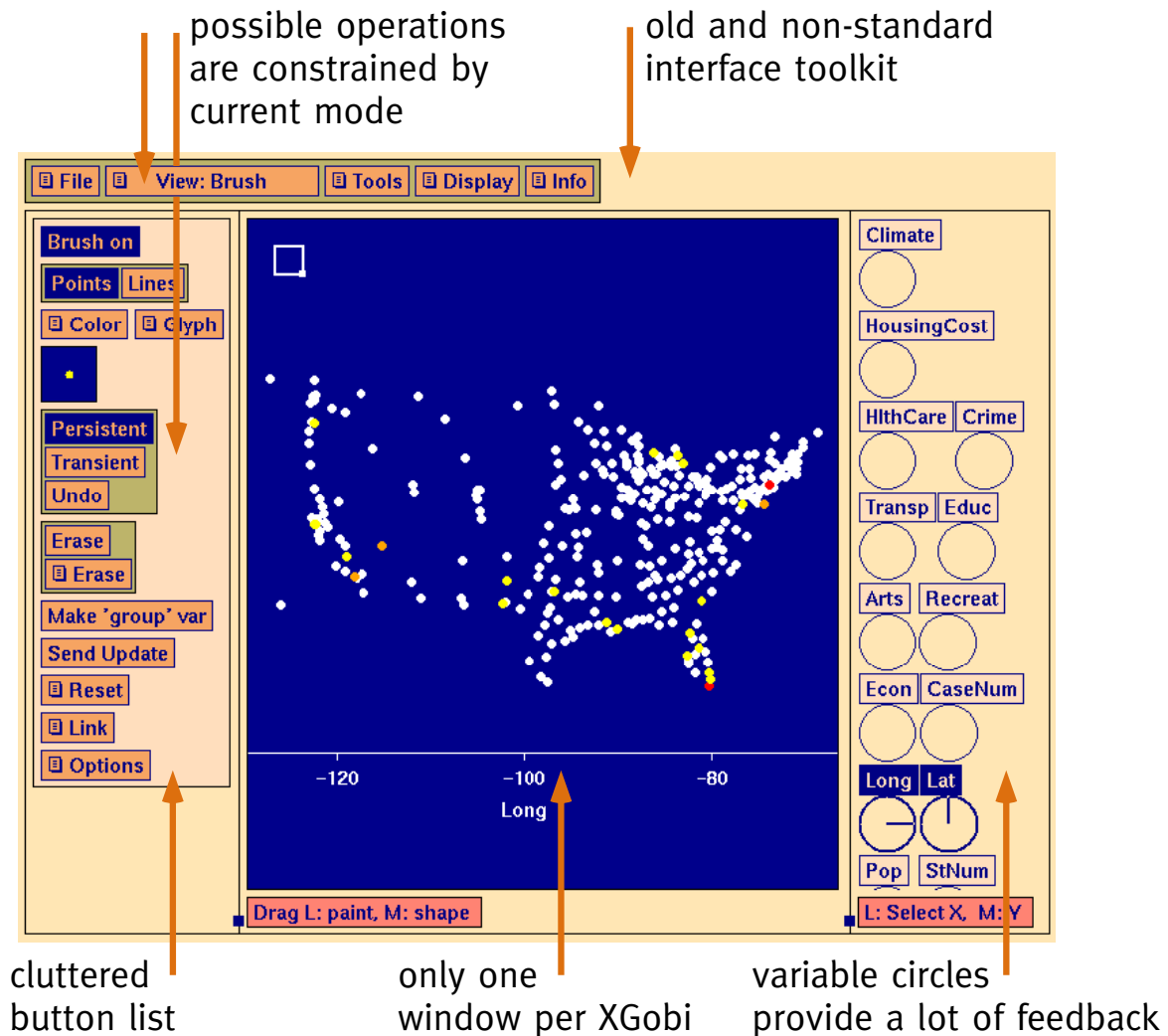


Figure 10: A typical XGobi window

8 Conclusion

When developing graphical user interfaces for interactive statistical graphics we should always keep in mind that “our best guess is not good enough”. An iterative optimization process with the potential user, or at least persons others than the implementers is indispensable in order to optimize the interface.

The examples of existing interactive statistical graphics software show that the complex tasks of direct manipulation interfaces can not be performed with trivial controls. We can not expect an optimal interface which will cope with all tasks. Most interfaces are optimized for a specific task.

Thus a “good enough” design might be the best we can achieve. This is not disappointing at all, because most software is still far away from being “good enough”, and there is still much room for improvement.

After giving this presentation at the Interface I had to answer the question why most of the examples of the hall of shame came from the most successful software company ever. A superficial answer is that people do not seem to care. In my opinion the true answer is — according to Donald A. Norman — that people stop complaining if they struggle with the interface of a tool — software in our case.

9 Digestif: The Lavatory Door

When flying from Munich to Chicago to the Interface Conference, I was presented with an empirical study of user interfaces. I was successful in trading seats to get more legroom, but now was facing the lavatory door of the 777 jet. This jetliner is certainly one of the most advanced airplanes of one of the most experienced aircraft manufactures.

The lavatory door has three exposed spots. The light, which indicates whether the lavatory is vacant or occupied is placed at the point where we usually expect a door handle, several inches below the ash tray, which is about the only thing that looks like a handle at the door, and — of course — the big sign saying “push” in the middle of the door at about 4 feet above the ground.

About 40% of all passengers grabbed for the ash tray. Another 40% tried to get hold of the light, and only a mere 20% pushed at the appropriate place and opened the door at their first attempt.

Obviously the door violated the mental model of a door of most passengers. Despite of their literacy they did not expect to face a folding door, but a normal door, which explains their failure. By the way, none of the passengers seemed to blame the designers of the door.

Acknowledgments

Many thanks go to Debby Swayne, who very much inspired me to give this talk. Parts of this talk are based on the manuscript of a talk Debby gave at Humboldt University of Berlin in Summer 1998. But more than that we argued a lot about user interfaces of software we wrote at AT&T Shannon Laboratories, which was always very inspiring.

My thanks also go to Steve Eick and Graham Wills, who invited me to give this talk and sponsored my trip.

References

Books and Papers:

Buja, A., Cook, D., Swayne, D. F. (1996), Interactive High-Dimensional Data Visualization, *Journal of Computational and Graphical Statistics*, Vol. 5, No. 1, 78–99.

Gentner, D., and Nielsen, J. (1996), “The Anti-Mac interface,” *Communications of the ACM* 39, 6 (August), 70–82.

Nielsen, J. (1993), *Usability Engineering* Academic Press, Boston.

Norman, D. A. (1988), *The Psychology of Everyday Things*, Basic Books.

Theus, M.; Hofmann, H. and Wilhelm, A.F.X. (1997), “Selection Sequences — Interactive Analysis of Massive Data Sets” *Computer Science and Statistics: Proceedings of the 29th Symposium on the Interface*, Springer, New York.

Theus, M. (1999), Mondrian — Interactive Statistical Graphics in JAVA. *Statistical Computing and Graphics Newsletter*, to be published.

Unwin, A.R., Hawkins, G., Hofmann H. and Siegl B. (1996), Interactive Graphics for Data Sets with Missing Values — MANET. *Journal of Computational and Graphical Statistics*, Vol. 4, No. 6.

Velleman, P. F. (1997), *Data Desk 6.0, Data Description*. Ithaca, New York.

Web Material:

Isys Information Architects Inc.
<http://www.iarchitect.com>

Jakob Nielson’s website on user interface design
<http://www.useit.com>

The ASA Statistical Graphics Library
<http://www.research.att.com/~dfs/videolibrary>