

**Applying a Theory of Graphical Presentation to the Graphic Design of
User Interfaces**

J. D. Mackinlay

UIR-R-1988-05

Applying a Theory of Graphical Presentation to the Graphic Design of User Interfaces

Jock Mackinlay
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304

Abstract: The increasing availability of computers with high-quality graphics and fonts has created an opportunity and an obligation for user interface designers. The opportunity is that designers can use graphical techniques to design more effective user interfaces. The obligation is that they must become experts at the design of graphical user interfaces. Current user interface toolkits provide very little design assistance. This paper describes a theory that supports automatic design of graphical presentations of relational information and shows how to extend it to support theory-driven design of graphical user interfaces.

"A picture worth a thousand words must first be a good picture" [Bow68]

1. Introduction

It has been observed that the goal of user interface design is to reduce the gulf between users and applications [Norm86] and that graphical techniques are a good way to achieve this goal [HHN86]. Given the power of human perception, many aspects of the user interface can be enhanced by the use of graphical techniques. Graphical techniques can improve the presentation of application or task information. For example, application information, such as program source texts, can be effectively presented with careful choice of fonts and layout [BM86], and task information,

such as program state, can be effectively presented with active graphical displays [Myers85]. Graphical techniques can also improve recall of application or task information [MC76] and make computer systems more satisfying and easier to learn [Schn87].

Unfortunately, it is not easy to design graphical user interfaces. Graphical interfaces can be worse than their non-graphical alternatives [WJLW85, Schn87]. The problem is that technological advances, such as hardware support for high resolution color and real time animation, have created an overwhelming space of design choices for the user interface designer. Without a deep understanding of how to make these design choices, designers will be forced to search this growing design space in an ad hoc manner, perhaps missing interface designs that will make computers easier for people to use.

The traditional solution to the problem of designing user interfaces has been to develop user interface toolkits, which are collections of pre-built interface modules and support software that foster the rapid prototyping of user interfaces. Unfortunately, user interface toolkits are only a partial solution to the problem of designing user interfaces. Rapid prototyping still requires design, and there is no guarantee that a toolkit will help the designer find an effective design. Furthermore, we still have the problem of designing the interface modules of the toolkit.

This paper describes research that begins to solve this problem with user interface toolkits by extending a theory of graphical presentation to the design of graphical user interfaces. The theory of graphical presentation was developed to automate the design of graphical presentations (such as bar charts, scatter plots, and connected graphs) of relational information

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and / or specific permission.

© 1988 ACM 0-89791-283-7 / 88 / 0010 / 0179 \$1.50

Marks: points, lines, and areas
Positional: 1D, 2D, and 3D
Temporal: animation
Retinal: color, shape, size, saturation, texture, and orientation

Figure 1. Bertin's graphical objects and their properties.

[Mack86a, Mack86b]. A fundamental assumption behind this theory is that graphical presentations are sentences of graphical languages, which are similar to other formal languages in that they have precise syntactic and semantic definitions. These graphical sentences are based on a common graphical vocabulary derived from graphic designer Jacques Bertin's analysis of the graphical techniques that are commonly used to encode information in presentation graphics (see Figure 1) [Bert83]. Bertin states that graphical presentations use graphical marks, such as points, lines, and areas, to encode information via their positional, temporal, and retinal properties.¹ The theory described in this paper explains how this vocabulary combines to form graphic designs and whether a given combination is effective. Thus, this theory of graphical presentation has two properties that support the automatic design of presentations: the ability to *generate* alternative designs and the ability to *test* these presentations.

The theory of graphical presentation is extended to the design of graphical user interfaces in two steps. The first step deals with the design of *interface presentations*, which are the static presentations of a user interface that do not involve any interaction or animation. The paper demonstrates that many applications can be described as collections of relational information, which means that the theory of graphical presentation can be used directly to design their interface presentations. The second step is to breathe interactive life into these static interface presentations. The paper describes how to accomplish this by modifying a graphics editor to understand the desired application functionality and the semantic properties of the interface presentation. Unfortunately, this theory-driven approach to interface design is limited. In particular, the paper

¹Temporal properties refer to the variation of one of the other properties over time. Retinal properties, such as color hue, are so called because the retina of the eye is sensitive to them, independent of the position of the object.

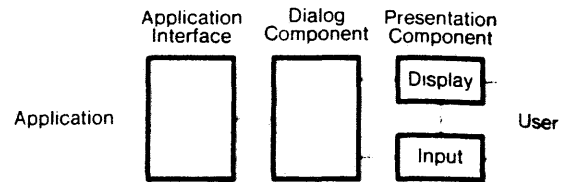


Figure 2. The Seeheim UIMS Model.

considers how the theory might be extended to handle highly-interactive applications that require user interfaces with animated presentations or carefully designed input techniques.

2. User Interface Toolkits

User interface toolkits are software tools that support the development of user interfaces. Figure 2 describes the runtime model for a group of user interface toolkits called User Interface Management Systems (UIMSs) [Pfaff85, Olsen87]. Besides runtime support, a user interface toolkit can support both the specification of user interfaces with a library of pre-built user interface modules and the analysis of running user interfaces with post-processing modules [TB85].

User interface toolkits support the design of graphical interfaces by promoting rapid prototyping and by encouraging graphic design efforts. A user interface toolkit promotes rapid prototyping with specification support and a library of interface modules. UIMS research, in particular, has focused on specification support for the dialog component shown in Figure 2. Rapid prototyping can lead to better designs by allowing the designer to explore a number of alternatives within the time and resource limits of a user interface design effort. A user interface toolkit encourages graphic design efforts because every graphic design improvement to its library of interface modules is also an improvement to application interfaces developed with the toolkit. Furthermore, applications based on the same toolkit are likely to use a consistent graphic design.

Unfortunately, toolkit design support is rather indirect. Consider using a toolkit to design a graphical interface for a straightforward class scheduling application that involves class prerequisites and the assignment of classes to quarters. The design effort involves choosing and combining interface modules from a potentially large library of interaction modules. For example, the designer of the class scheduling user interface might

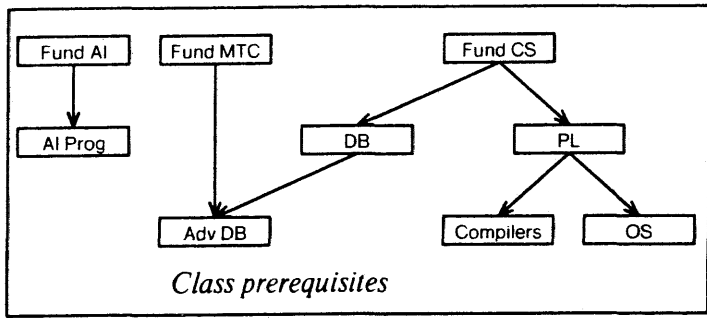


Figure 3. A graph describing the prerequisites among computer science classes.

choose a "grapher" module for the class prerequisites (see Figure 3) and a "table" module for the assignment of classes to quarters (see Figure 4). The problem is that the designer needs to know if these are good choices and if they can be combined to form an effective interface. In fact, such design decisions are also a problem for the developer of a user interface toolkit. What should the library contain? Are there design guidelines for choosing modules? How should interface modules be designed for combination? The essence of the difficulty with the current toolkit approach to interface design is the lack of a theoretical understanding about the design of graphical user interfaces. Given such an understanding, we would know what modules to put in the toolkit library, when to choose them, and how to combine them. Ultimately, a theory-driven approach to user interface design will be developed that supports software tools that help directly with the design of graphical user interfaces.

A number of other research efforts have demonstrated that the careful analysis of graphical presentations can lead to software tools for graphic design. Feiner's APEX system automatically generates a sequence of images that describe actions in a 3D world consisting of several cabinets of a sonar system [Fein85]. The system carefully tailors the sequence of images to omit irrelevant or redundant details. The graphic design issues surrounding the merging of icons and 3D images are also considered. Weitzman's DESIGNER system evaluates the graphic design of user interface panels for the Steamer system, which is used to instruct engineering students [Weit86]. Myers's PERIDOT system for constructing direct-manipulation interfaces by demonstration uses heuristic rules to adjust graphical objects to conform to each other [MyBu86].

Quarter			
Fall 85	Fund CS	Fund MTC	
Winter 86	Adv DB	PL	
Spring 86	OS	DB	Fund AI
Fall 86	Compilers	AI Prog	

Class schedule

Figure 4. A table describing the scheduling of computer science classes.

3. Theory-driven Design of Graphical User Interfaces

The research described in this paper is an initial effort to develop a theory-driven approach to the design of graphical user interfaces by creating a formal description of the mappings between applications and the perceptible aspects of their interfaces. Ciccarelli used a similar approach when he successfully described a wide range of user interfaces with the model described in Figure 5 [Cicc84]. He used two databases to describe the endpoints of the mapping, one representing the application and the other representing the presentation. The application database represents information in a form useful for an application, such as relations or procedure calls. The presentation database represents information in a form useful to a user, such as graphical objects or input actions. The mappings between these databases are maintained by presenter and recognizer processes. The notion of mappings between representations has also been used to develop a successful UIMS that allows such mappings to be specified with object-oriented techniques [SHB86].

Although useful as a descriptive technique, Ciccarelli's model does not allow us to generate and test alternative designs for graphical user interfaces. The additional step taken here is to describe these mappings in linguistic terms as a semantic relationship between the graphical vocabulary of the interface and the application information. Alternative designs are generated by composing elements of the graphical vocabulary. These alternatives are tested by evaluating their

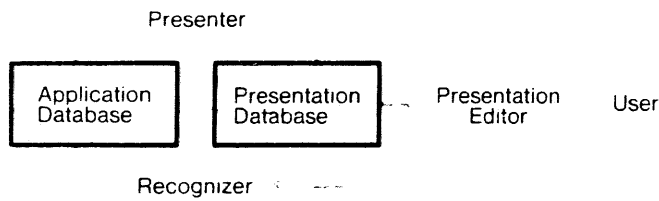


Figure 5. Ciccarelli's Primitive Presentation System Model used to describe a wide range of user interfaces. The absence of an arc from the presentation database to the user reflects his focus on modeling input.

expressiveness and effectiveness. The result is a design space that can be searched by generating alternatives design and then testing them.

Given the overwhelming complexity of the mappings between applications and the perceptual aspects of their interfaces, it is natural to start any theoretical accounting of the design of such mappings by making simplifying assumptions that focus the research. In particular, the next section starts by describing research that is focused on the design of static presentations of application information. This research includes the development of a system called APT (A Presentation Tool) that automates the design of two dimensional (2D) static presentations (such as bar charts, scatter plots, and connected graphics) of relational information. Given this foundation, section 5 describes how a graphics editor can breathe interactive life into these static presentations. Although quite useful, the resulting interfaces are not the final answer in interface design. Section 6 discusses open problems associated with the design of highly-interactive interfaces.

4. Designing Presentations of Application Information

4.1 Graphical Languages

All communication is based on the fact that the participants share conventions that determine how messages are constructed and interpreted. For graphical communication these conventions indicate how arrangements of graphical objects encode information. The theory-driven approach formalizes these conventions by taking the view that graphical presentations are actually sentences of graphical languages that have precise syntactic and semantic definitions. A mathematically precise development of this idea is beyond the scope of this paper and can be found elsewhere [Mack86a, Mack86b]. However,

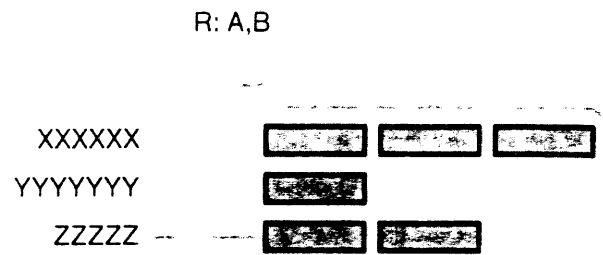


Figure 6. The Encodes relationships for the vertical position language given an abstract relation $R: A,B$. The gray lines indicate that the domain sets of the relation are encoded by the objects of the graphical sentence and the tuples of the relation are encoded by the relative positions of the marks on the axis.

an intuitive understanding can be formed by considering the vertical axis presentation for the computer science class schedule shown in Figure 4. The syntactic definition of the corresponding vertical axis language identifies well-formed graphical sentences consisting of points positioned on a vertical axis. The semantic definition identifies the correspondences between these graphical objects and properties and the encoded information. In particular, the points in Figure 4 encode computer science classes, the axis encodes an ordered set of class quarters, and the position of the points on the axis encodes the quarter in which a class is scheduled. More formally, the Encodes relation is used to describe these correspondences symbolically. For example, the Encodes relations for Figure 4 are:

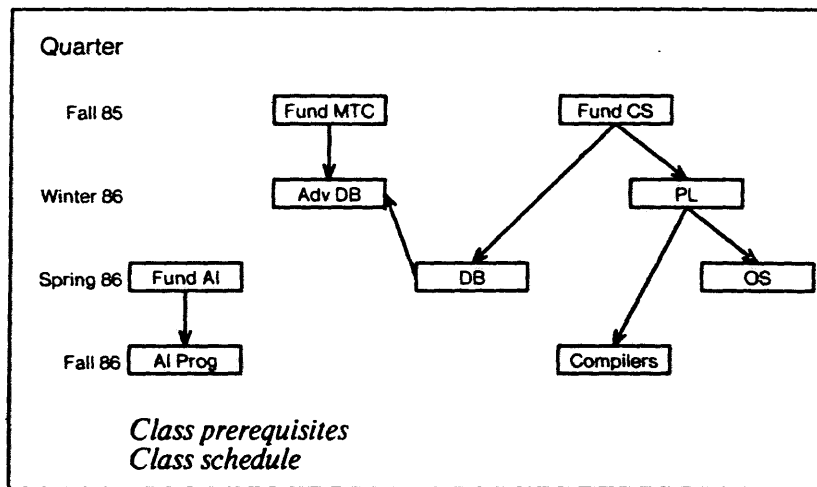
Encodes(Points,Classes)
 Encodes(Axis,Quarters)
 Encodes(Position(Points,Axis),
 Schedule(Classes,Quarters)),

where the first argument is a graphical object or property and the second argument is the encoded application information. The abstract Encodes relations for the vertical single position language are described graphically in Figure 6.

A careful analysis of the graphical language definitions commonly used in presentation graphics leads to two results: (1) a composition algebra for describing a space of alternative graphical designs, and (2) criteria for evaluating these alternatives. Given these results, it is possible to develop a synthesis algorithm that automatically designs a wide range of graphical presentations of relational information.

Encoding Technique	Primitive Graphical Language
Single Position	Horizontal axis, Vertical axis
Apposed Position	Line chart*, Bar chart, Plot chart
Retinal List	Color, Shape, Size, Saturation, Texture*, Orientation*
Map	Road map*, Topographic map*
Connection	Tree, Acyclic graph, Network*
Misc. (Angle, Contain, . . .)	Pie chart*, Venn diagram*, . . .

Figure 7. A basis set of primitive graphical languages. The star (*) indicates primitive languages not implemented by APT's rendering code.



APT

Figure 8. Composite presentation for the Prerequisite and Schedule relations. Note that Advanced Databases is scheduled before its prerequisite.

4.2 Composition Algebra

A *composition algebra* consists of a basis set of primitive graphical languages that capture fundamental graphical techniques for encoding information and some composition operators that can form these techniques into complex presentations. Figure 7 contains the primitive languages used in the APT research. APT's composition operators are based on the following principle:

Compose two designs by merging parts that encode the same information the same way.

For example, consider the network presentation shown in Figure 3 of the prerequisites for some computer science classes, which has the following Encodes relations:

Encodes(Points,Classes)
 Encodes(Link(Points,Points),
 Prerequisites(Classes,Classes)).

This design can be composed with the vertical axis presentation of the class schedule because they both use point objects to encode the same set, that is,

Encodes(Points,Classes). The resulting presentation is shown in Figure 8, which makes it clear that the advanced database class is scheduled before its prerequisite. These three presentations were generated by the APT prototype.

4.3 Evaluation Criteria

Evaluation criteria for the graphic design of relational presentations can be based on many different factors. For example, a design can be judged to be effective when it can be interpreted accurately or quickly, when it has visual impact, or when it can be rendered in a cost-effective manner. The APT research concentrates on generating designs that can be accurately interpreted. Dealing with multiple, perhaps conflicting, effectiveness criteria is an open research problem. Two types of evaluation criteria were developed for APT: (1) *expressiveness* and (2) *effectiveness*.

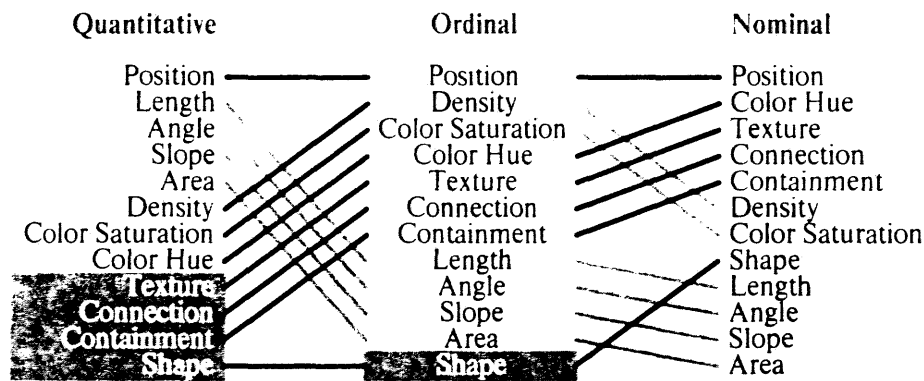


Figure 9. Ranking of perceptual tasks. The columns are for three different types of information. Tasks higher in the chart are perceived more accurately than tasks lower in the chart. The tasks shown in gray are not relevant to that type of information.

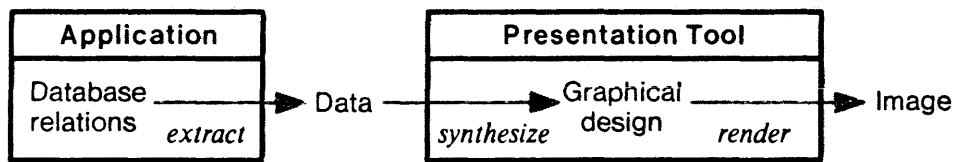


Figure 10. Model combining an application and a presentation tool.

An *expressiveness* criterion, which is derived from a precise language definition, is associated with each primitive graphical language:

A graphical language can be used to present some information when it can express *exactly* the input information, that is, all the information and only the information.

Expressing additional information is potentially dangerous because it may not be correct [MG85].

Unlike expressiveness, which only depends on the syntax and semantics of the graphical language, *effectiveness* also depends on human perceptual capabilities, which are not well understood. Therefore, one must conjecture effectiveness criteria that are both intuitively motivated and consistent with current empirically verified knowledge about human perceptual capabilities. Cleveland and McGill made the key observation that was used to conjecture APT's expressiveness criteria, namely: people accomplish the perceptual tasks associated with the interpretation of graphical presentations with different degrees of accuracy [CM84,Clev80]. They identified and ranked the major tasks associated with the presentation of quantitative information and experimentally verified their ranking. Given that APT must also present non-quantitative information, I extended this ranking as shown in Figure 9; the

modified ranking has yet to be experimentally verified.

4.4 The Synthesis Algorithm Used in APT

The theory of graphical presentation sketched above can be used both constructively to generate designs and analytically to test alternatives. As such, it can be embedded in a tool as a computational theory of design. As a demonstration of this, the theory of graphical presentation has been embedded in the APT prototype, which automatically designs graphical presentations of relational information. Figure 10 illustrates how a presentation tool can be combined with an application to present information to a user. The application extracts some information from its database (perhaps using statistical analysis). The presentation tool then synthesizes a graphical design and renders an image that presents this information. Given the ability to generate and test, standard AI techniques were used to implement APT. In particular, APT is based on a depth-first backtracking search algorithm (see Figure 11), which involved the following three steps:

- 1) *Partitioning*. The information to be presented is divided into partitions that satisfy the expressiveness criteria of at least one of the primitive languages.

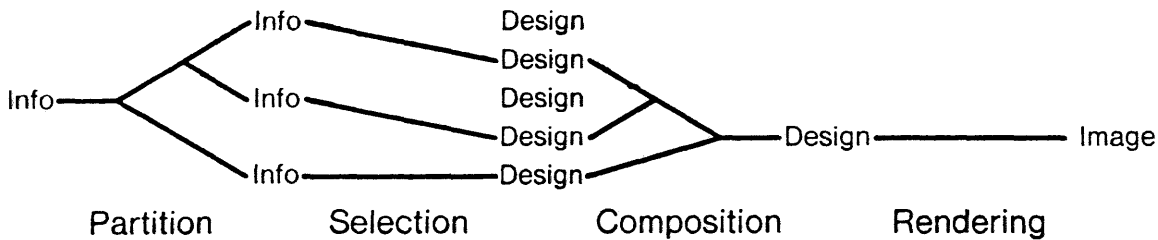


Figure 11. APT's search architecture. Gray lines indicate decisions reversed by backtracking.

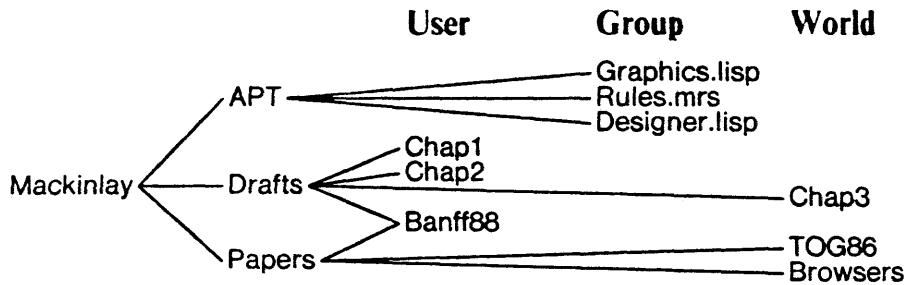


Figure 12. A proposed design for a file browser. The top level directory has three subdirectories and the horizontal position of the files indicates their protections. File protection can be changed by moving nodes from one column to another.

- 2) *Selection.* Given a partition, use the evaluation criteria to select a primitive language.
- 3) *Composition.* Composition operators are used to compose individual designs into a unified presentation of all the information.

Backtracking occurs when partitions cannot be matched to primitive graphical languages or when partial presentations cannot be composed together.

4.5 Designing Interface Presentations

The theory of graphical presentation used to develop APT focused on the graphical presentation of relational information. Fortunately, application information is often relational, which means that the theory of graphical presentation can be used directly to design interface presentations for applications. For example, consider the problem of designing an interface presentation for a file browser application that includes information about the hierarchical structure of the files in the user's directory and about the allowed file access to these files (that is, user, group, or world). This application information can be formally described as a binary relation from a nominal set (of files and directories) to itself and a function from that nominal set to an ordered set (of

protection values). This formal description has the same basic structure as the class schedule relations presented in Figure 8, which immediately suggests the interface presentation shown in Figure 12. In this case, the tree has been rendered horizontally to take advantage of the horizontal shape of file names. This design makes it easy to see the directory structure and the file access information simultaneously. For example, the file Chap3 in the Drafts directory has mistakenly been given world access.

The theory of graphical presentation described above also allows us to compare this design with the more traditional listings of files. UNIX file structures allow files to be members of more than one directory. APT's expressiveness criteria indicates that traditional file listings do not show such information, but that node and link diagrams do. For example, Figure 12 indicates that the Drafts and Papers directories share one file, Banff88.

Although we have only considered application information so far, it turns out that interface presentations for application commands can also be designed using the theory of graphical presentation. Given that the nominal, ordinal, and quantitative properties of information are central to the theory of graphical presentation, it is natural to analyze application commands for similar properties. Generally, application commands have a nominal

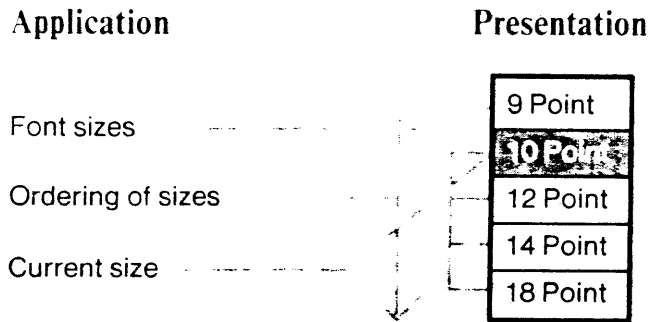


Figure 13. The mappings between an application and a simple menu. The gray lines indicate the mappings.

structure, often grouped into a small number of sets that separate various types of commands from each other. In this case, the primary design goal is to make sure that the user can easily identify these command types. Interface presentations such as control panels and pull-down menus have successfully used position and containment to present nominal application commands, which is consistent with the ranking in Figure 9 for nominal information. However, Figure 9 also indicates that color and texture should be used to present nominal application commands when hardware advances in high-resolution color displays make these techniques practical.

In some cases application commands can have an ordinal or even quantitative structure. For example, consider setting font sizes in a typical personal computer text editor. Figure 13 describes both a typical menu for font size commands and the formal analysis of the relationship between this interface presentation and the corresponding application. The rectangle objects encode the alternative font sizes, the vertical ordering of the rectangles encodes the ordering of the point sizes, and the saturation of the rectangles encode which point size is currently selected.

The theory of graphical presentation can be used to generate and evaluate alternative designs for presenting application commands. One problem that often occurs in user interface design is that the designer has one set of conventions in mind while the user has a different set. For example, the only menu in a typical personal computer text editor that is likely to be ordered is the font size menu, which means the user may miss the ordering convention and search font size linearly each time a font size needs to be chosen. Such problems can be solved by using a composition algebra to generate alternative designs

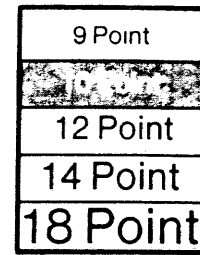


Figure 14. Using label size to encode the ordering of the font sizes.

that indicate the structural relationships in a more effective way. For example, the designer might improve the design for the font size menu by searching for a different graphic technique to encode the font size ordering and composing it with the basic menu design. In this case, the obvious choice is to use the size of the rectangle's label to encode the font size. The resulting design is shown in Figure 14. However, this design will not work when the menu must include large point sizes whose combined height would make the menu larger than the vertical height of the display. If this can happen, the designer can reject the label size technique and move on to other graphical techniques, such as encoding the font size by a line length that can be placed horizontally in the corresponding menu rectangle.

Evaluation criteria could also be used to design graphical presentations that are tailored to particular output devices. For example, the ranking in Figure 9 indicates that color is more effective than saturation for nominal information, such as whether a menu item is currently selected.

5. Turning Graphical Presentations into User Interfaces

The theory of graphical presentation focuses on the graphic design of static presentations of application information. User interfaces, however, must support active presentations that allow a user to manipulate interface presentations and affect the corresponding applications. This section describes how to modify a graphics editor to turn static presentations into interactive interfaces. The key observation is that the graphics editor must allow precisely the graphical manipulations that make sense given the encoding relationships that have been designed between a proposed interface presentation and its application.

Users communicate with applications that have

graphical interfaces by manipulating graphical objects and properties of the interface. Thus, the basic architecture of a graphical interface will include both a component that supports the user's graphical manipulation and a component that translates the result of a manipulation into requests that can be communicated to the application. However, not all possible manipulations correspond to application requests, and not all possible requests will be granted by the application. An effective approach is to have the manipulation component support a range of manipulation functionality but restrict the user in any given situation to manipulations that are acceptable to the application [Lieb85].

The graphical properties described in Figure 9 specify a good range of manipulation functionality for interface presentations designed with the theory of graphical presentation. It turns out that a graphics structure editor with standard capabilities, such as affine transformations, filled objects, and object subparts, has this range of functionality. For example, a position manipulation is a translation, a texture manipulation is a change in the fill of an object, and a shape manipulation is a change in object subparts.

In general, there are three reasons to restrict a user's manipulations: 1) it must be possible to translate a manipulation into application terms, 2) translated requests should match the functionality of the application, and 3) specific requests should pass application error checking. The first two restrictions can be accomplished by making the editor understand the semantic properties of interface presentations and simple specifications of the supported application functionality. The third restriction requires feedback from the application.

First, the graphics structure editor must prohibit user manipulations that cannot be translated to application requests. For example, consider position manipulations of nodes in the class scheduling presentation in Figure 8. The translation component of the graphical interface can only generate update requests to the application if the vertical position of the manipulation is restricted to the quarter axis. In general, this can be accomplished by making the graphics structure editor understand the encoding relationships that are used by the graphical theory of presentation to design interface presentations. Fortunately, this is not too much work because the set of primitive relationships (shown in Figure 7) is quite small.

Second, the graphics structure editor must

prohibit user manipulations that do not match the desired functionality of the application. For example, the implementor of the hypothetical class scheduling application may not want users to change class prerequisites, which means the graphics editor must not allow the deletion, movement, or addition of link objects. This can be accomplished by making the graphical structure editor understand additional specifications that indicate which encoding relations map to adjustable application values.

Third, the graphics structure editor must prohibit user manipulations that do not produce acceptable application requests. For example, the implementor of the class scheduling application may be willing to let faculty edit the prerequisites as long as a cycle of prerequisites are not generated. In general, the structure editor cannot support this type of restriction without a detailed understanding of the application's functionality. Given that the structure editor is intended to support a wide range of applications, it is unlikely that it will be able to understand all the resulting details. This problem is, in fact, shared by user interface toolkits in general. The standard solution is to ask the application to check potential updates, either during the graphical manipulation if the check does not slow down the manipulation feedback, or in a commit step after a manipulation is terminated. For example, if the class schedule application checks during the manipulation of link object, the user will not be allowed to add any link that causes a cycle of prerequisites. If the application checks after the manipulation, the user will draw the link cycle, but an error message will be produced and the manipulation will be undone.

6. Limitations of the Current Theory-driven Approach

The theory-driven approach to user interface design described in this paper is appropriate for applications that only need static interface presentations augmented by conventional input techniques that might be provided by a graphics structure editor. Highly-interactive applications, on the other hand, require user interfaces that have animated presentations or carefully designed input techniques. This section discusses how these issues might be addressed.

Many applications involve rapid state changes that require the user interface to use animation techniques. Typically, such animations describe these state changes by the temporal variation of one of the

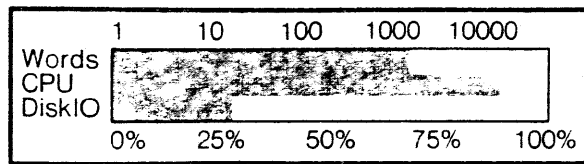


Figure 15. An animated bar chart describing word allocation, CPU load, and disk load.

graphical properties described in Figure 9. For example, the Cedar programming environment animates the bar chart shown in Figure 15 to display the current system load [SZBH87]. The lengths of the bars vary to indicate changes in word allocation, CPU load, and disk IO requests.

Although the theory of graphical presentation describes the graphical properties that might be varied by an animation, it does not address many of the issues that are involved in designing such variations. For example, the theory needs a formalization of the duration and degree of variation so that their semantics can be determined. Once such a formulation has been developed, the next problem is to deal with the various graphic design problems associated with developing animated presentations. For example, given the speed of computers, animations of application state can often occur too quickly to be directly perceptible. If this happens, the design of the animated presentation must be changed. For example, the actual bar chart animation corresponding to Figure 15 has been designed to avoid such problems by the careful choice of the intervals used to sample the state changes and the use of averaging functions to smooth out variations in the animations. Another issue is how to combine animations to form an effective user interface. Unrestricted combination might easily lead to an interface that confuses rather than informs.

In addition, highly-interactive applications often require specialized input techniques that allow the user to respond to the application quickly. For example, Hill has developed a user interface toolkit for building two-handed interfaces [Hill86]. I believe that the theory of graphical presentation described in this paper can be extended to support the design of input techniques because input design is the dual of output design. Output design involves the relationship between application information and the graphical properties of a user interface. Input design, on the other hand, involves the relationship between application functionality and input devices. The

exploitation of this duality promises to be a fruitful direction for further research.

7. Conclusion

As we have seen, the graphic design of user interfaces is a difficult problem that requires methods and tools that are based on a theoretical understanding of user interfaces. The emerging theory, described in this paper, represents a modest step towards such an understanding.

The existence of a theory that supports the automatic design of graphical presentations of relational information provides reasonable evidence for the possibility of theory-driven user interface design. Theory-driven user interface design should not be viewed as an effort to supplant the role of human creativity in interface design. In fact, its goal is exactly the opposite: a theory and artifacts built upon it can serve as amplifiers to creativity. After all, user interfaces are designed for people, and our requirements and expectations will be forever evolving. Perhaps the ultimate lesson of this research should be that creativity and theory go hand in hand, one to inspire and initiate and the other to refine, test, and extend.

Acknowledgments

I would like to thank Stuart Card and Polle Zellweger for their helpful discussions that improved this paper. The comments of the anonymous referees were also appreciated.

References

- [Bert83] J. Bertin. *Semiology of Graphics*. The University of Wisconsin Press, 1983. Translated by William J. Berg.
- [Bow68] W. Bowman. *Graphic Communication*. John Wiley & Sons, NY, 1969.
- [BM86] R. Baecker and A. Marcus. Design Principles for the Enhanced Presentation of Computer Program Source Text. *Proceedings CHI'86*, pages 51-58, April 1986.
- [Cicc84] E. C. Ciccarelli. *Presentation Based User Interfaces*. PhD thesis, MIT, August 1984. Also MIT technical report AI-TR-795.
- [Clev80] W. S. Cleveland. *The Elements of Graphing Data*. Wadsworth Advanced Books and Software, Monterey, California 93940, 1980.
- [CM84] W. S. Cleveland and R. McGill. Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods. *Journal of the American Statistical Association* 79(387), pages 531-554, September 1984.
- [Fein85] S. Feiner. APEX: An Experiment in the Automated Creation of Pictorial Explanations. *IEEE Computer Graphics*

- and Applications*, 5(11), pages 29-37, November 1985.
- [HHN86] E. L. Hutchins, J. D. Hollan, and D. A. Norman. Direct Manipulation Interfaces. In D. A. Norman and S. W. Draper, editors. *User Centered System Design*, pages 87-124. Lawrence Erlbaum, 1986.
- [Hill86] R. D. Hill. Supporting Concurrency, Communication, and Synchronization in Human-Computer Interaction - The Sassafras UIMS. *ACM Transactions on Graphics* 5(3), pages 179-210, July 1986.
- [Lieb85] H. Lieberman. There's More to Menu Systems Than Meets the Screen. *Computer Graphics* 19(3), SIGGRAPH'85 Proceedings, pages 181-189, July 1985.
- [Mack86a] J. Mackinlay. *Automatic Design of Graphical Presentations*. PhD thesis, Stanford University, 1986.
- [Mack86b] J. Mackinlay. Automating the Design of Graphical Presentations of Relational Information. *ACM Transaction on Graphics* 5(2), pages 110-141, April 1986.
- [MG85] J. Mackinlay and M. R. Genesereth. Expressiveness and Language Choice. *Data and Knowledge Engineering* 1(1), pages 17-29, June 1985.
- [MC76] D. Markoff and D. Carcel. *Total Recall*. Grosset and Dunlap, NY, 1976.
- [MyBu86] B. A. Myers and W. Buxton. Creating Highly-Interactive and Graphical User Interfaces by Demonstration. *Computer Graphics* 20(4), SIGGRAPH'86 Proceedings, pages 249-258, August 1986.
- [Myers85] B. A. Myers. The Importance of Percent-Done Progress Indicators for Computer-Human Interaction. *Proceedings CHI'85*, pages 11-17, April 1985.
- [Norm86] D. A. Norman. Cognitive Engineering. In D. A. Norman and S. W. Draper, editors. *User Centered System Design*, pages 31-61. Lawrence Erlbaum, 1986.
- [Schn87] B. Shneiderman. *Designing the User Interface*. Addison-Wesley, 1987.
- [SHB86] J. L. Sibert, W. D. Hurley, and T. W. Bleser. An Object-Oriented User Interface Management System. *Computer Graphics* 20(4), SIGGRAPH'86 Proceedings, pages 259-267, August 1986.
- [Olsen87] D. R. Olsen, et al. ACM SIGGRAPH Workshop on Software Tools for User Interface Management. *Computer Graphics* 21(2), pages 71-147, April 1987.
- [Pfaff85] G. E. Pfaff. *User Interface Management Systems*. Springer-Verlag, 1985.
- [SZBH86] D. Swinehart, P. Zellweger, R. Beach, and R. Hagmann. A Structural View of the Cedar Programming Environment. *ACM Transactions on Programming Languages and Systems* 8(4), pages 419-490, October 1986.
- [TB85] P. P. Tanner and W. A. S. Buxton. Some Issues in Future UIMS Development. G. E. Pfaff, editor, *User Interface Management Systems*, pages 67-79. Springer-Verlag, 1985.
- [Weit86] L. Weitzman. *DESIGNER: A Knowledge-Based Graphic Design Assistant*. Institute for Cognitive Science Technical Report 8609, La Jolla, CA, 92093, July 1986.
- [WJLW85] J. Whiteside, S. Jones, P. S. Levy, and D. Wixon. User Performance with Command, Menu, and Iconic Interfaces. *Proceedings CHI'85*, pages 185-191, April 1985.