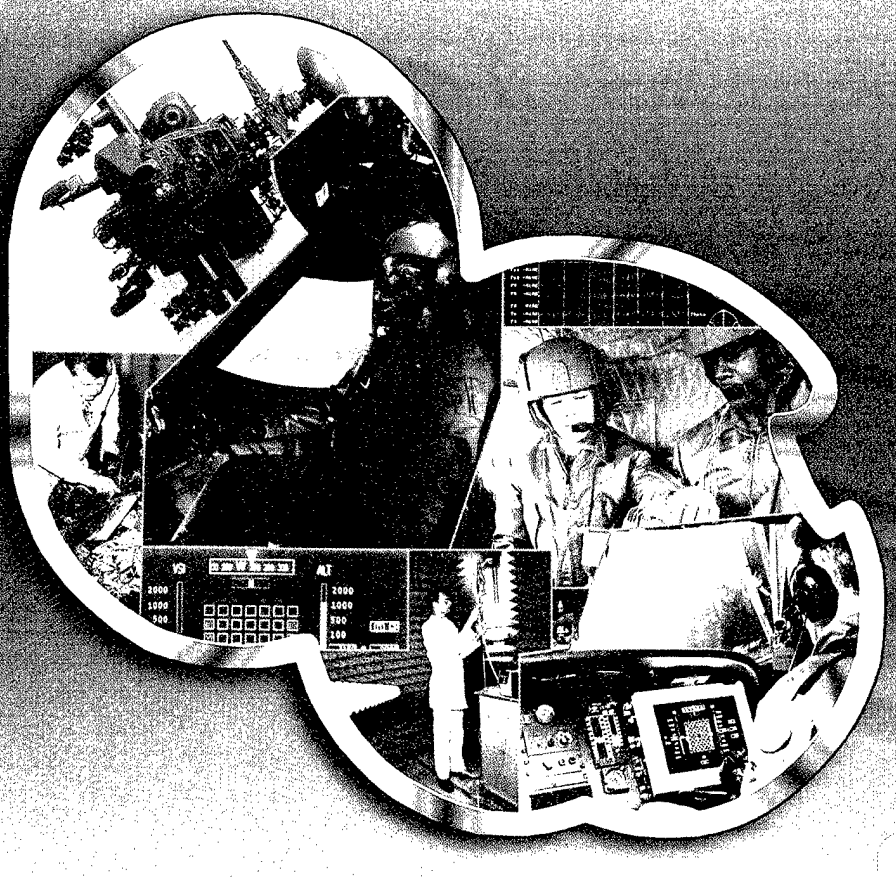
USAARL Report No. 2002-22

MFDTool (Version 1.3): A Software Tool for Optimizing Hierarchical Information on Multifunction Displays

by Gregory Francis and Clarence E. Rash



Aircrew Health and Performance Division

August 2002

USAARL

U.S. Army
Aeromedical Research
Laboratory

Notice

Qualified requesters

Qualified requesters may obtain copies from the Defense Technical Information Center (DTIC), Cameron Station, Alexandria, Virginia 22314. Orders will be expedited if placed through the librarian or other person designated to request documents from DTIC.

Change of address

Organizations receiving reports from the U.S. Army Aeromedical Research Laboratory on automatic mailing lists should confirm correct address when corresponding about laboratory reports.

Disposition

Destroy this document when it is no longer needed. Do not return it to the originator.

Disclaimer

The views, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other official documentation. Citation of trade names in this report does not constitute an official Department of the Army endorsement or approval of the use of such commercial items.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION
Unclassified

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION

2b. DECLASSIFICATION / DOWNGRADING

3. DISTRIBUTION / AVAILABILITY OF REPORT
Approved for public release, distribution unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
USAARL Report No. 2002-22

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
U.S. Army Aeromedical Research Laboratory

6b. OFFICE SYMBOL (If
MCMR-UAD

7a. NAME OF MONITORING ORGANIZATION
U.S. Army Medical Research and Materiel Command

6c. ADDRESS (City, State, and ZIP Code)
P.O. Box 620577
Fort Rucker, AL 36362-0577

7b. ADDRESS (City, State, and ZIP Code)
504 Scott Street
Fort Detrick, MD 21702-5012

8a. NAME OF FUNDING / SPONSORING ORGANIZATION

8b. OFFICE SYMBOL (If

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS (City, State, and ZIP Code)

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
62787A	3M162787A879	DX	178

11. TITLE (Include Security Classification)
(U) MFDTool (Version 1.3): A software tool for optimizing hierarchial information on multifunction displays

12. PERSONAL AUTHOR(S)
Gregory Francis, Clarence E. Rash

13a. TYPE OF REPORT	13b. TIME COVERED FROM	TO	14. DATE OF REPORT (Year, Month,	15. PAGE COUNT
Final			2002 August	65

16. SUPPLEMENTAL NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	cockpit design, hierarchy, multifunction displays (MFDs), workload
01	02		
24	07		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report provides the description of an enhanced version (1.3) software program called MFDTool. MFDTool allows human computer interaction experts to optimize the design of multifunction display (MFD) screens. However, the original version of MFDTool was limited to only one type of interaction, and was thereby restricted to only a small subset of possible MFDs. MFDTool, Version 1.3, provides for multiple types of interactions, greatly enhancing its ability to define appropriate MFD designs. In addition to providing immediate benefit to MFD designers, MFDTool allows for the systematic investigation of which types of interactions are important in human-computer interactions with MFDs. Previously, such studies were limited because there was no means of insuring that an MFD design was appropriately built to accommodate a particular type of interaction. With MFDTool, such designs can be generated easily, and this will allow experimentation on those designs to reveal important aspects of the human-computer interaction.

20. DISTRIBUTION / AVAILABILITY OF
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL
Chief, Science Support Center

22b. TELEPHONE (Include Area
(334) 255-6907

22c. OFFICE SYMBOL
MCMR-UAX-SS

DD Form 1473, JUN 86 *Previous editions are obsolete.*

Table of contents

List of figures

Introduction

This report describes an extensive modification of a software program called MFDTool. MFDTool allows human computer interaction experts to optimize the design of multifunction displays (MFDs). The use of MFDTool is described in the MFDTool designer's guide (Appendix), which is available for download at http://www.psych.purdue.edu/~gfrancis/MFDTool. While the designer's guide discusses how to use MFDTool, it gives only a cursory description of the theoretical ideas behind MFDTool. This report fills in that description by explaining the definition and purpose of multiple interactions with an MFD.

Issues in MFD design

As computers are integrated into military cockpits, the human factors issues of interfacing with a multifunction display become extremely important. Some of the human factors issues of MFD design have been reviewed previously (Francis and Reardon, 1997; Francis, 1998, 1999; Reardon and Francis, 1999). The design of the MFD is important because the effective use, learning, and tendency to make errors are all related to the design (Rogers et al., 1996; Obradovich and Woods, 1996; Cook and Woods, 1996; Cuomo et al., 1998; Reising and Curry, 1987; Dohme, 1995; Sirevaag et al., 1993).

One of the findings of human factors is that the specific details of the human-computer system have a significant impact on determining the appropriate design. While the human factors field has a number of important guidelines for the development of MFDs, the application of those guidelines depends on the details of the particular situation. This means that previously created MFD designs offer only partial guidance for the development of new MFD designs. Thus, for example, an MFD that was appropriate for a fixed wing aircraft may be inadequate for a rotary wing aircraft, and vice-versa. Even worse, the underlying principles that guided the design of one MFD system may not apply to another.

Since each MFD must be built nearly from scratch, anything that can speed up the process will be useful. One useful tool is the VAPs design tool that is marketed by Virtual Prototypes, Inc. VAPs provides a graphical user interface for rapid prototyping of multifunction displays. This speeds the development of MFD designs, thereby allowing a designer to consider a larger number of possible designs. However, with VAPs, the decisions of how to construct the MFD still depend on the designer's experience and intuition.

MFDTool addresses a complementary problem by focusing on a particular subtask of MFD design, namely, how page labels in an information hierarchy should be assigned to hardware buttons. MFDTool asks the designer to provide an information hierarchy, the properties of the MFD hardware, and aspects of the human-computer interaction that the designer believes are important. MFDTool quantifies this information so that an optimization can be performed that builds the best MFD design subject to these MFD properties.

Although an MFD may be casually described to have one type of human-computer interaction (e.g., pressing of bezel buttons), in reality, multiple interactions are involved in almost every system. For example, to press bezel buttons, a user needs to be able to see the buttons and reach them. Thus, both the visual display of information (a type of visual interaction) and the motor

commands to reach buttons (a type of motor interaction) are involved. These different interactions have different properties, and the MFD design should consider those differences. As another example, in many instances, successful use of the MFD depends on the user being able to remember where certain information is located. A memory interaction may be related to other types of interactions, but may also introduce its own idiosyncrasies.

More generally, a single MFD may be used in multiple ways and by multiple users. For example, in a military cockpit, a pilot may have multiple means of interacting with the MFD. Bezel buttons around the MFD are a common type of touch-interaction. A hand-on-throttle control is often another means of interacting with the MFD. The MFD should be designed so that the pilot can quickly and efficiently use either type of interaction to retrieve information from the MFD. Additionally, a copilot may also have multiple interactions, but the frequency of using the hand-on-throttle interaction may be lower than for the pilot. An MFD designer should consider all of these factors and build an MFD that is optimized to accommodate all types of interactions by different people.

Thus, the issue is how to consider all of these different types of interactions and users. The current design methods involve prototyping, testing, and further revision. This approach does tend to satisfy all the different interactions and their needs. However, this satisfaction is achieved with the cost of considerable time and expense. Moreover, because the design process tends to stop when everything feels "right," there is no way that a rapid prototyping process can insure that a good enough design is actually the best that could be constructed. Francis and Reardon (1997) suggested that MFD design needed to be quantified in order to meet the goals of the design process in a rigorous way. Toward that end, they identified a set of quantitative constraint costs. Different MFD designs produce different cost values. With this quantitative measure of the "goodness" of the design, it was possible to apply standard optimization techniques to find an MFD design that minimized the constraints costs.

However, the constraint costs identified by Francis and Reardon (1997) were limited in scope and applicability. An elaboration of these ideas was proposed by Francis (1999), who created a software tool, called MFDTool, that allowed designers to specify different types of constraints to fit their intuitions about what was important in the design process. However, that version of MFDTool was also limited to only one type of interaction, and was thereby restricted to only a small subset of possible MFDs. This documents describes MFDTool version 1.3, which includes a systematic way of describing interactions and constraints relative to those interactions. This allows for a quantitative description of MFD design that can accommodate multiple interactions with the MFD and multiple users of the MFD. Optimization relative to the constraints can then produce an MFD design that is the best across an extremely complex set of requirements. What is fundamentally new is the role that multiple interactions play in the development of MFD design.

Interactions

In using an MFD, the primary task of the user is to retrieve needed information. Exactly how the user retrieves the information depends on the particular details of the MFD and the user. For example, a novice user may need to read individual labels to identify which ones will lead to the

information he/she requires. Such a user will interact with the MFD with a visual scan to find and read labels and then reach with his/her hand to press buttons. On the other hand, a highly experienced user may have the required button pushes memorized and only needs to execute movements between buttons to reach desired information. An intermediately skilled user may sometimes execute memorized movements and other times may need to search an MFD screen to find a necessary label.

If it were known that all the users would be highly trained experts, designing an MFD would be much easier. One could assume that all users would receive sufficient practice to memorize where every piece of information was in the MFD structure, and the MFD design could be focused on minimizing the time needed to move between buttons to reach the desired information in the most efficient manner. Likewise, if it were known that all users would be novices there would probably be little reason to minimize movement time (unless it also helped visual search processes). Instead, the design would focus on promoting a reasonable arrangement of information to promote the search for information. For an intermediate user, a designer might wish to minimize movement time, on the assumption that the user will have some sequences memorized. At the same time, the designer might want to insure that the user's hand does not block labels that must be moved to, in case the user needs to read the labels. Unfortunately, although the field of human factors knows a good deal about the properties of human computer interactions, there is no known way to *predict* whether a user will be acting as a novice or an experienced user during a particular search for a particular piece of information. As a result, a complete quantitative description of human-computer interaction is not currently feasible for most situations.

What is feasible, however, is for the designer to identify what types of users will be using the MFD and to identify interactions that correspond to different user types. These interactions can be described as relationships between pairs of buttons. For example, during visual search, focus on one button will leave other buttons in the periphery, where visual acuity is poor. As a result, if a user must search the MFD screen for a needed label, there may be an advantage to having the most frequently searched for label located on the same button that was just pressed. Rarely used items could be placed on farther away buttons that may be in the periphery of the visual field.

The preceding example supposes that visual search will be faster if an eye movement is not required. This may be true in some situations. However, such an approach may be difficult for users to work with. Perhaps a simpler design is to associate frequently used labels with certain buttons (e.g., start on the upper left). This type of consistency might be easier because the user would always know where to look to find the most frequently used labels. This description of the search process is also a type of interaction. It can be described as a relationship between buttons. Namely, after each button press, it will be easier for the user to find items at some buttons than at other buttons.

Without knowledge of the details of the MFD and the users, there is no way to know which (if either) of the above search styles is appropriate. Whatever choice is made, it can be described as a relationship between buttons. Indeed, many different types of interactions can largely be described as a relationship between buttons. The relationship can identify how difficult it is to

go from one button to another, represent a distance (broadly defined) between buttons, or simply describe the relative positions of buttons.

Thus, in MFDTool, each type of human-computer interaction is defined by a set of *interaction coefficients*; each coefficient represents a relationship between a pair of buttons. For example, a Euclidean distance interaction would consist of interaction coefficients that identify the Euclidean distance between each pair of buttons. As another example, some MFDs have the user push arrow keys to "tab" through buttons. The interaction coefficients here might indicate the number of pushes that are required to go from one button to another.

Interaction coefficients need not be true distances in the mathematical sense. For example, in some situations, it may be difficult to go from the left side of the MFD to the right side, but easy to go the other way (e.g., perhaps the hand occludes the buttons and labels on the right side). Although it breaks down for some interactions, a designer can often think of the interaction coefficients as identifying how difficult it is for a user to go from one button to another. Larger numbers indicate more difficulty. The task of designing an MFD is then to assign page labels to buttons to minimize cost functions that are defined relative to the interaction coefficients.

In general, a designer can specify the interaction coefficients to be whatever is desired. For an MFD design crafted for a particular individual, multiple interactions can be used to describe the way this user will interact with the MFD. One interaction may describe the movement time between buttons, taking into account the size of the user's hand, reach of fingers, and speed of movements. Another interaction might take into account the ability of this user to search for labels, taking into account aspects of the user's ability to identify targets in his/her peripheral vision. Still another interaction might describe an entirely different type of user interface (e.g., hand-on-throttle control). After defining these interactions, appropriate cost functions could be defined, and the MFD could be optimized for that particular individual.

On the other hand, an MFD may be used by multiple individuals. To satisfy all of these users, the designer could define separate sets of interactions for each individual user; or could combine data across individuals and define interactions for different sets of users (e.g., small, medium, or large hands).

It may be a difficult task in some instances to specify the interaction coefficients for a particular MFD. It requires an understanding of the human-computer interface and perhaps the creation of a model that describes how the user(s) will interact with the interface. In principal, this type of analysis should be carried out for any human-computer interface design. MFDTool provides an easy means of calculating five sets of interaction coefficients that correspond to some types of human-computer interactions. The designer can also provide MFDTool with arbitrary types of interaction coefficients. These are described below.

Fitts' movement

A common form of interaction with an MFD is finger pointing, where a user presses buttons by moving his finger (or perhaps a mouse or pointer pen) from one button to another. There is a model of how long it takes people to move a pointer over a given distance to a target of a given

size. MFDTool uses a form of Fitts' Law (Fitts, 1954) that says that the movement time, *I[i,j]* between button *i* and button *j* is:

$$I[i,j] = I_m \log_2 \left(\frac{2H_{ij}}{S_j} + 1 \right).$$

Here, H_{ij} is the center-to-center Euclidean distance between the buttons; S_j is the size of the second button (defined by default as the minimum of button height and width), $\log_2$ is the logarithm in base 2, and I_m is a parameter with units of milliseconds (ms) per bit. I_m is empirically measured, and, for finger movements, values between 70 and 120 ms/bit are common. MFDTool uses I_m=100 ms/bit.

Constraints on this interaction can be set, for example, to minimize the time needed to move between buttons while retrieving information from the MFD. It is surely easier to press the same button four times than to move back and forth across the MFD screen to press a sequence of buttons. A designer may want to insure that frequently used page labels are assigned to buttons so that reaching that page label involves repeated selection of a common button.

Euclidean distance

This set of interaction coefficients describes the physical distance between button centers. The interaction coefficient for two buttons *i* and *j*, centered on (x_i,y_i) and (x_j,y_j), respectively, is computed as

$$I[i,j] = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

This distance is useful whenever a physical relationship is called for. A designer may want to insure that related labels are in physically similar locations. This might allow for visual grouping and help the development of memory if the same type of page label is in a consistent location across different pages.

City-block distance

This set of interaction coefficients describes the city-block distance between button centers. The interaction coefficient for two buttons *i* and *j* centered on (x_i,y_i) and (x_j,y_j), respectively, is computed as

$$I[i,j] = |x_i - x_j| + |y_i - y_j|,$$

where | | indicates absolute value.

City-block distance might be useful as an intermediate step in the calculation of some types of interactions. For example, an interface that involves tabbing through menu options may be a variation of a city-block distance.

X-directed

This set of interaction coefficients describes the horizontal distance and direction of button centers. The interaction coefficient for two buttons *i* and *j* centered on (x_i,y_i) and (x_j,y_j), respectively, is computed as

$$I[i,j] = x_j - x_i.$$

Position *(0,0)* corresponds to the upper left corner of the MFD screen. A positive interaction coefficient indicates that the second button is to the right of the first. A negative coefficient indicates that the second button is to the left of the first.

This coefficient does not really indicate difficulty of movement between a pair of buttons, but the absolute value of this coefficient would indicate the horizontal distance between buttons. *X*-directed is most useful for situations where the designer wants to constrain directional relationships between labels. For example, a designer could specify that all labels named *Checking* should be on a button that is to the left of the previous button press. This might introduce a type of "motor" memory that would make learning to use the MFD easier.

Y-directed

This set of interaction coefficients describes the vertical distance and direction of button centers. The interaction coefficient for two buttons i and j centered on (x_i,y_i) and (x_j,y_j), respectively, is computed as

$$I[i,j] = y_i - y_j.$$

Position *(0,0)* corresponds to the upper left corner of the MFD screen. A positive interaction coefficient indicates that the second button is above the first. A negative coefficient indicates that the second button is below the first. *Y*-directed has pretty much the same use as *X*-directed.

Arbitrarily defined

In addition to the automatically calculated interactions described above, a designer can identify any arbitrary set of interaction coefficients and use them as an interaction. This could be done, for example, in a spreadsheet, and then the results copied and pasted into MFDTool.

Constraints

Once an interaction is defined, MFDTool allows the user to identify six types of constraints, which can be mixed and matched as desired. Each constraint can be imposed on any defined interaction.

- **Global difficulty:** If one ignores shortcuts and backward links, then moving through the hierarchy from top to bottom can be described by a single finite sequence of button presses. When the designer specifies the frequencies of search for different pieces of MFD information, MFDTool associates page labels with buttons in a way to minimize the average difficulty needed to reach information.

- **Pages to close buttons:** Often labels on a single screen are related to each other and the designer wants the related page labels to be grouped together on nearby buttons. At other times, labels on different MFD screens are related and the designer wants those labels to be associated to the same or nearby buttons (e.g., *CANCEL* should be in the same place

on every page). MFDTool allows the designer to specify as many of these constraints as desired.

- **Pages to fixed buttons:** Sometimes a designer wants to restrict a single label or multiple labels (either on the same screen or different screens) to a subset of the possible buttons (e.g., always put left engine information on the left side of the MFD screen). MFDTool allows the designer to specify as many of these constraints as desired.

- **Path difficulty:** The use of some MFDs requires users to retrieve certain combinations of information. If a user has to first check the status of one system, then the status of a second, and then the status of a third, there will be a path of visited pages that correspond to this combination of information searches. Moreover, because the system information may be scattered across the MFD hierarchy, designers often include hyperlinks, or shortcuts, to the top of the hierarchy or to other MFD hierarchy locations. MFDTool allows the designer to identify these paths and then MFDTool optimizes the assignment of page labels to buttons so as to minimize the difficulty of executing these button sequences. MFDTool allows the designer to specify as many of these paths as desired.

- **Pages to far buttons:** Sometimes a designer may want a set of labels to be separated as much as possible. This could be useful to insure that a user does not confuse similar labels that are functionally different. It could also help the designer organize labels into different positions on the screen.

- **Parent to child variability:** This constraint allows a designer to impose certain types of consistency on the layout design. This constraint attempts to minimize, among specified pages, the variability (standard deviation) in interaction coefficients between the buttons assigned to a page's parent and to itself. For example, if Euclidean distance is used, then specified pages will be assigned to buttons so that clicking on the button of a page's parent will then require a move of a constant distance to reach the desired page's button. This would likely provide a consistent "motor memory" that might make learning and using the MFD easier.

In MFDTool, each constraint has a corresponding numerical cost function that measures how well a constraint is being satisfied by the current MFD design. Smaller cost values correspond to better designs relative to that constraint. An optimization algorithm searches through a variety of MFD designs to find one that minimizes (or nearly so) the sum of costs.

An example of MFD design

To demonstrate the capabilities of MFDTool, consider the design of a relatively simple system for entering alphabet characters. The Qwerty keyboard is the standard for two-handed keying of information. For other systems such as palm pilots and one-handed keyboards in aircraft, there is no established standard. The default design tends to be to order the letters alphabetically. This is probably suboptimal, as some letters are used more frequently than others and some letter

strings are used more often than other letter strings. A hard coded keyboard is a simple MFD (with only one function assigned to each key), and MFDTool can be applied to such a system.

Textware Solutions (http://www.twsolutions.com) recently proposed the Fitaly keyboard, which they claimed was able to increase typing by one finger/pen to almost as fast as a Qwerty two-handed keyboard. A schematic of the Fitaly keyboard is shown in Figure 1.

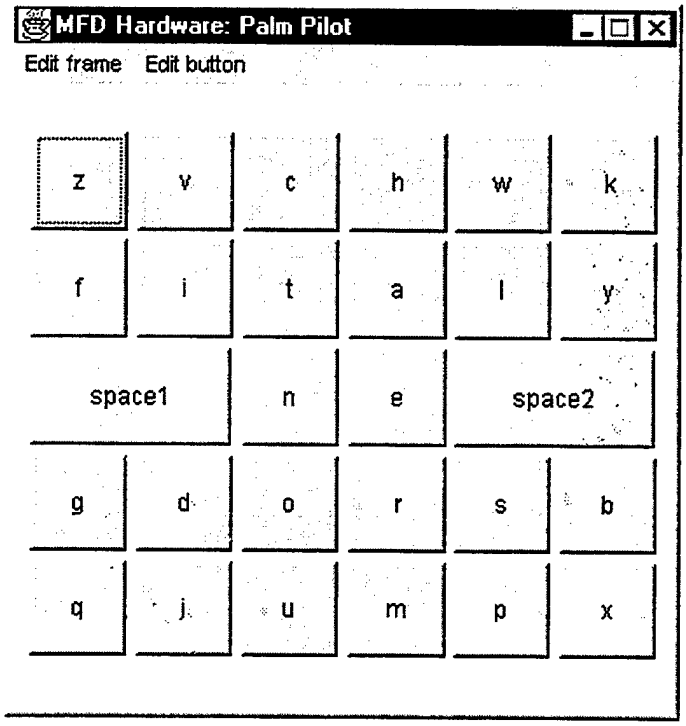


Figure 1. The original Fitaly keyboard for one-finger use.

The engineers at Textware Solutions based their design on consideration of the frequency of using individual letters and the frequency of letter-to-letter transitions. The former frequencies are published on their web site, and were used to create an analogous design in MFDTool. Since users will soon memorize where every letter is on the keyboard, the layout of letters should be so as to minimize movements. A Fitt's Law interaction was used. The Fitaly keyboard includes two larger than usual keys for spaces, and the MFDTool design was similarly restricted so that these keys could only be used for space characters. Interestingly, MFDTool creates a design quite similar to the Fitaly design, see Figure 2. In particular, the six letters in the middle three rows of the middle two columns are the same in both designs. These correspond to the most frequently used letters.

The discrepancies between the two designs probably correspond to the effect of letter-to-letter transitions, which the current MFDTool example does not consider (but could be implemented with multiple (*Path difficulty* constraints). Without consideration of those transitions, the MFDTool design is slightly more optimized relative to the Fitt's Law interaction. The MFDTool

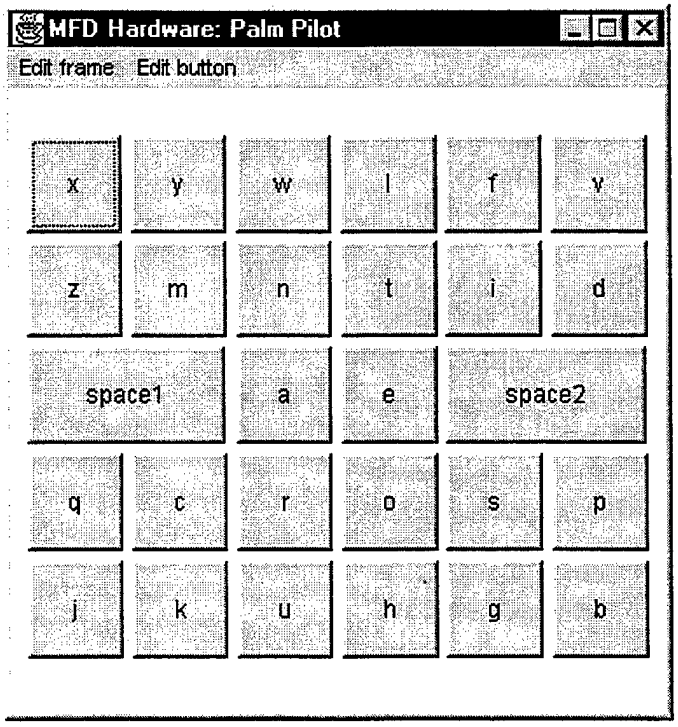


Figure 2. The optimal one-finger keyboard designed by MFDTool with the Fitaly hardware design.

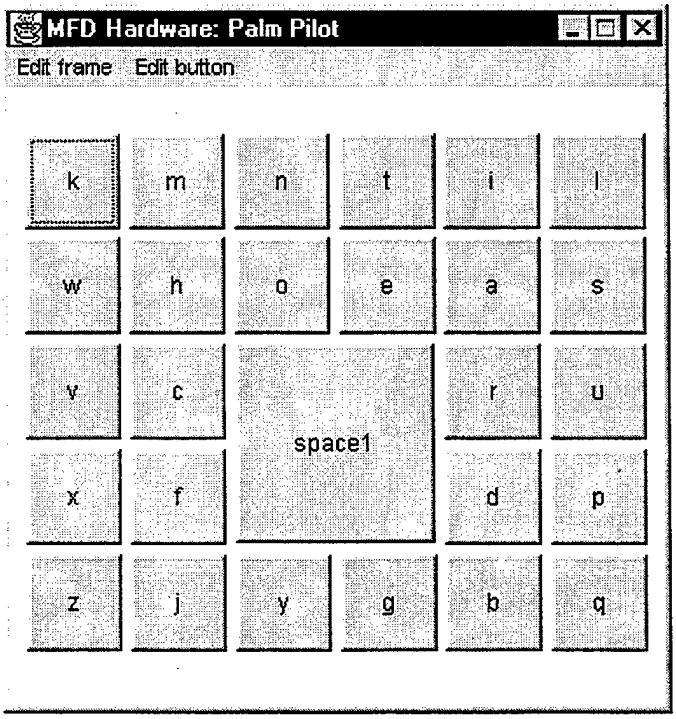


Figure 3. A variation on the Fitaly design, with one space button near the middle of the keyboard.

design has a global constraint cost of 182.77, while the Fitaly design has a global constraint cost of 185.85.

MFDTool can also be used to consider optimization of other hardware designs. For example, in Figure 3, the two space buttons are merged into a single large button and placed slightly off the center of the screen, the optimized assignment has a larger average Fitt's movement time (184.28) than the optimized layout for the hardware used by the original Fitaly system.

If the space buttons are merged and placed on the lower left as in Figure 4, the optimized design produces a global constraint cost of 181.35. This is a difference of 1.5 ms faster than the optimal design with the original Fitaly hardware configuration. It is 4.5 ms faster than the Fitaly layout. A 4.5 ms difference is small, but it accumulates over extended use. MFDTool also identifies that the "home" for the user should be at the *e* button.

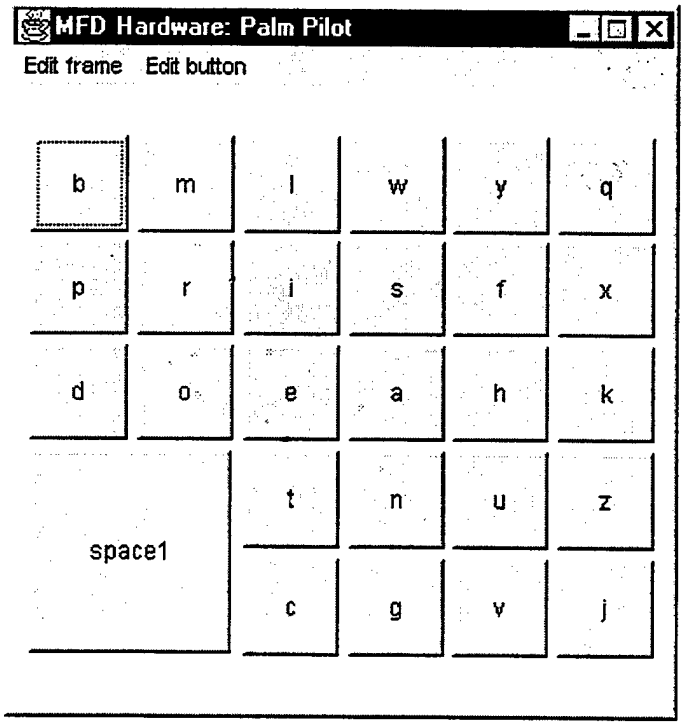


Figure 4: A variation of the Fitaly design with the space buttons merged and placed on the lower left.

Even better results are found for the design in Figure 5, where the space button is one row high and four columns wide and placed on the middle bottom of the display. The Fitt's Law interaction's global constraint for the optimized design is 167.61 ms, which is 13.74 ms faster than any other design discussed so far. Relative to the original Fitaly layout, this design has an 18.24 ms advantage.

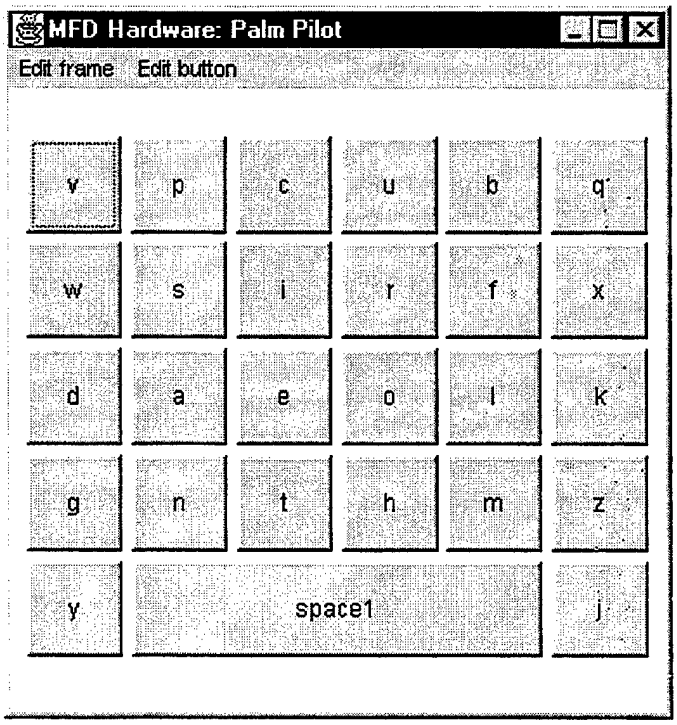


Figure 5: A variation of the Fitaly design with the space buttons merged and placed on the bottom row. This optimized design results in the smallest average movement time between letters.

Of course, these alternatives to the original Fitaly design may not be as good as the original if the transition probabilities between letters are also considered. Also, one would want to experimentally validate the predicted differences in the designs.

An even better approach for the use of MFDTool would be to identify the specific text that must be entered for particular uses of the keyboard. Then create a set of *path difficulty* constraints that corresponds exactly to that text. Such an arrangement will automatically include the relevant statistics on frequency of letter use and will include statistics for higher order combinations of letter strings. The optimization of the keyboard can then be specified for each use of the keyboard (or for keyboards that need to be used in multiple situations or by multiple users).

Conclusions

Version 1.3 of MFDTool allows for multiple types of interactions, and this greatly enhances its ability to define appropriate MFD designs. In addition to providing immediate benefit to MFD designers, MFDTool (version 1.3) also will allow for the systematic investigation of which types of interactions are important in human-computer interactions with MFDs. Previously, such studies were limited because there was no means of insuring that an MFD design was appropriately built to accommodate a particular type of interaction. With MFDTool (version

1.3), such designs can be generated easily, and this will allow experimentation on those designs to reveal important aspects of the human-computer interaction.

References

Cook, R. and Woods, D. 1996. Adapting to new technology in the operating room. Human Factors. 38: 593-613.

Cuomo, D. L., Borghesani, L., Khan, K. and Violett, D. 1998. Navigating the company web. Ergonomics in Design, 6, 7-14.

Dohme, J. 1995. The military quest for flight training effectiveness. Vertical Flight Training. W. Larsen, R. Randle, and L. Popish (Eds.) NASA Reference Publication 1373.

Fitts, P. M. 1954. The information capacity of the human motor system in controlling the amplitude of movement. Journal of Experimental Psychology. 47: 381-391.

Francis, G. 1998. Designing optimal hierarchies for information retrieval with multifunction displays Fort Rucker, AL: U.S. Army Aeromedical Research Laboratory. USAARL Report No. 98-33.

Francis, G. 1999. A software tool for the design of multifunction displays Fort Rucker, AL: U.S. Army Aeromedical Research Laboratory. USAARL Report No. 99-20.

Francis, G. and Reardon, M. 1997. Aircraft multifunction display and control systems: A new quantitative human factors design method for organizing functions and display contents Fort Rucker, AL: U.S. Army Aeromedical Research Laboratory. USAARL Report No. 97-18.

Obradovich, J. H. and Woods, D. D. 1996. Users as designers: How people cope with poor HCI design in computer-based medical devices. Human Factors. 38: 574-592.

Reardon, M. J. and Francis, G. 1999. Reducing the risk of aviator-multifunction display interface problems with human factor models and optimization design methods. SAFE Journal. 29, 100-106.

Reising, J., and Curry, D. 1987. A comparison of voice and multifunction controls: Logic design is the key. Ergonomics. 30: 1063-1077.

Rogers, W., Cabrera, E., Walker, N., Gilbert, K., and Fisk, A. 1996. A survey of automatic teller machine usage across the adult life span. Human Factors. 38: 156-166.

Sirevaag, E., Kramer, A., Wickens, C., Reisweber, M., Strayer, D., and Grenell, J. 1993. Assessment of pilot performance and mental workload in rotary wing aircraft. Ergonomics. 36: 1121-1140.

Appendix

MFDTool 1.3 designer's guide.

MFDTool: A software aid for the design of multifunction displays

Designer's Guide

Version 1.3

Gregory Francis[1]
Purdue University
1364 Psychological Sciences Building
West Lafayette, IN 47907
gfrancis@psych.purdue.edu
http://www.psych.purdue.edu/~gfrancis/home.html
and
U.S. Army Aeromedical Research Laboratory
Ft. Rucker, AL 36362-0577

19 February 2001

[1]The views, opinions, and/or findings contained in this report are those of the author and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

This work was supported by the Army Aeromedical Research Laboratory (Dr. Kent Kimball) under the auspices of the U.S. Army Research Office Scientific Services Program administered by Battelle (Delivery Order 547, Contract No. DAAL03-86-D-0001).

Abstract

This document is a guide to MFDTool, a software aid for the design of multifunction displays (MFDs). MFDTool applies an optimization algorithm to designer-specified constraints thereby creating the best layout of MFD information for MFD hardware. The guide specifies the types of MFD situations where MFDTool applies and describes the steps needed to define constraints and start the optimization approach. A sample MFD design problem (involving an automated teller machine) is discussed.

MFDTool 1.2 varies from earlier versions in that it now provides a means of specifying multiple types of interactions with the MFD. This allows a designer to consider a multitude of different types of human-computer interfaces and optimize over all of them. Additional changes include a new interface, easier definition of constraints, user control of optimization parameters, editing of MFD hierarchies, editing of MFD hardware properties, and manual creation of MFD designs.

MFDTool 1.3 includes a slightly enhanced user interface, minor bug fixes, and an installation program for W95/98/NT.

Contents

1 Installation

MFDTool is written in the Java programming language, which allows it to be run on any machine with a Java Runtime Environment (JRE). JREs are available for nearly every modern operating system; so, in principle, MFDTool should run almost anywhere. MFDTool does not come with a JRE, but one can be freely downloaded from `http://www.javasoft.com`. You will need to download either the Java Development Kit (JDK), sometimes also called the Standard Development Kit (SDK) or the Java Runtime Environment (JRE). Either option will allow you to run the compiled code. (Although many web browsers include a JRE, they cannot run MFDTool. MFDTool must read from and write to the computer's hard drive, and the JREs in web browsers prohibit such actions.)

After installing the JRE, you can install MFDTool (you can also reverse the order). The mechanisms for doing this vary depending on your operating system. The following is believed to be correct, but it may vary somewhat as operating systems change. In particular, many operating systems are planning to allow Java programs to start with a double-click from the user interface.

The following instructions are only for the installation of MFDTool. You need to install the JRE on your own.

1.1 Windows 95/98/NT

Double click on the `MFDTool1.3setup.exe` icon to start automatic installation of MFDTool1.3.

To run MFDTool, select the MFDTool option from the Start menu.

If for some reason the installation program should not work, you can install MFDTool by hand using the instructions for UNIX/Linux systems. To run MFDTool you can then go to the MFDTool directory and double click on the `MFDTool.pif` file.

1.2 UNIX/Linux

Uncompress the file `MFDTool1.3.tar.gz`. Move the folder MFDTool to where ever you want to install MFDTool.

To run MFDTool, open a terminal to the MFDTool directory and type `java MFDTool`. Certain UNIX systems allow you to double click on Java class files to make them run. If so, just double click on the file `MFDTool.class`.

1.3 Macintosh

Installation is the same as for UNIX/Linux. You can use Stuffit to uncompress the tar.gz file.

The following is based on the Sun web site's description of how to run a Java program on MacOS. This does not include MacOS X, which may be different.

To run MFDTool, open the Tools folder, then open the Application Builders folder. Finally, open the JBindery folder, which contains the application JBinder. Drag and drop the `MFDTool.class` file in the MFDTool folder on top of the JBindery icon. You should now see a dialog box. Click on `Run`, and MFDTool should start.

2 Purpose

MFDTool is software that helps a multifunction display (MFD) designer optimize assignment of MFD information to MFD hardware commands (e.g., button pushes). To understand how MFDTool can help in the design process; the following section describes some of the tasks involved in MFD design.

2.1 Some difficulties of MFD design

Many computer devices for information display involve fixed hardware switches and flexible software pages. These types of devices are called multifunction displays (MFDs) or multi-purpose displays. The information in these devices is often arranged hierarchically so that a user starts at a top level and moves down the hierarchy by selecting appropriate MFD pages. Often the hardware devices are real or simulated buttons that remain in fixed positions. Common examples of MFDs include automated teller machines, pagers, aircraft cockpit display panels, and various medical devices. As the user moves through the hierarchy, the MFD screen changes, thereby providing information to the user. The creation of effective MFDs is a difficult task, and designers must decide how many buttons to include, the hierarchical arrangement of information, and the mapping of page labels to hardware (typically buttons). MFDTool is a computer program that helps in the last of these tasks.

Before discussing what MFDTool does, it may be useful to discuss what it does *not* do. MFDTool cannot specify MFD hardware. Decisions on MFD hardware are influenced by a variety of factors, including environmental conditions, cost of devices, previous versions of the MFD, and other details that may be particular to an MFD's specific purpose. In general, there is no method for *a priori* designing the MFD hardware, although there are a variety of guidelines. Often times a designer is simply given a standard hardware setup and told to put the information "in" the device. MFDTool also does not select what types of information should be included in the MFD. The selection of information in an MFD and its hierarchical arrangement is a task that largely must be determined by a human designer because it involves identifying what types of information are necessary, and creating labels that are meaningful for the user and the MFD purpose. No computer or algorithm, currently, can identify the semantic and organizational relationships between MFD information.

However, at some point in the design of an MFD, decisions must be made about how to map the various parts of the information hierarchy to user actions (e.g., button pushes). Mapping hierarchy information to MFD buttons is a challenging task. The human-computer interactions involved in accessing information from an MFD are complicated and not entirely understood. Moreover, even a small hierarchy database can be mapped to hardware buttons

in a vast number of ways, so combinatorial explosion quickly precludes an exhaustive search of all possible mappings. Therefore such mappings are, at best, created by experts who rely on experience and general guidelines like the following:

1. Frequently used functions should be the most accessible.
2. Time critical functions should be the most accessible.
3. Frequently used and time critical functions should be activated by the buttons that feel "ideally located" (e.g., top of a column of buttons).
4. Program repeated selection of the same button. Failing that, program sequential selections to adjacent buttons.
5. The number of levels in the hierarchy should be as small as possible.
6. The overall time to reach functions should be minimized.
7. Functions that are used together should be grouped on the same or adjacent pages.
8. Related functions on separate pages should be in a consistent location.
9. Related functions should be listed next to each other when on a single page.
10. Consider the types of errors crew members might make and place functions accordingly to minimize the effect of those errors.

While many of these guidelines correctly identify the key characteristics of good MFD design, application of these criteria is problematic because they often conflict with each other. For example, should a frequently used function be placed by itself near the top of the hierarchy (1) or should it be placed in a submenu with its related, but infrequently used, functions (7)? Likewise, should criteria (3), (4) or (7) dominate selection of a button for a specific function? Currently, there is no quantitative method of measuring the tradeoffs and designers try out different options until the whole system "feels" good. This is a time consuming task because movement of a single function can require other changes throughout the MFD. As a result, MFD creation largely remains an artistic endeavor, depending primarily on the experience, intuition, and hard work of the designer.

The best artists use good tools to help them in their craft. MFDTool handles part of the complexity of measuring the impact of various guidelines. MFDTool cannot build an MFD from scratch, and the design will still depend on the experience and effort of the designer. MFDTool does allow the designer to consider a larger range of possibilities by automating part of the design process, thereby freeing the designer to focus on other tasks.

2.2 How MFDTool helps

MFDTool focuses on a subset of the guidelines identified above that can be recast in terms of an optimization problem. MFDTool requires that the designer has a specified MFD hardware system that describes the sizes and positions of MFD buttons (the approach can be modified to other types of interfaces, but buttons are a common interface type). MFDTool also requires that the designer specifies the hierarchical arrangement of information pages in the database. MFDTool also allows for hyperlinks that move back up the hierarchy (e.g., `RETURN`) or function as shortcuts, as described below.

Given this information, MFDTool allows the user to identify six types of constraints, which can be mixed and matched as desired. Each constraint can be imposed on different defined interactions (discussed below).

1. **Global difficulty:** If one ignores shortcuts and backward links, then moving through the hierarchy from top to bottom can be described by a single finite sequence of button presses. When the designer specifies the frequencies of search for different pieces of MFD information, MFDTool associates page labels with buttons in a way to minimize the average difficulty needed to reach information. This constraint corresponds to guidelines (1), (6), and often (4), above.

2. **Pages to close buttons:** Often labels on a single screen are related to each other and the designer wants the related page labels to be grouped together on nearby buttons. At other times labels on different MFD screens are related and the designer wants those labels to be associated to the same or nearby buttons (e.g., `CANCEL` should be in the same place on every page). MFDTool allows the designer to specify as many of these constraints as desired. This constraint corresponds to guidelines (8), and (9), above.

3. **Pages to fixed buttons:** Sometimes a designer wants to restrict a single label or multiple labels (either on the same screen or different screens) to a subset of the possible buttons (e.g., always put left engine information on the left side of the MFD screen). MFDTool allows the designer to specify as many of these constraints as desired. This constraint accommodates guideline (3) above, but also allows for more general restrictions.

4. **Path difficulty:** The use of some MFDs requires users to retrieve certain combinations of information. If a user has to first check the status of one system, then the status of a second, and then the status of a third, there will be a path of visited pages that correspond to this combination of information searches. Moreover, because the system information may be scattered across the MFD hierarchy, designers often include hyperlinks, or shortcuts, to the top of the hierarchy or to other MFD hierarchy locations. MFDTool allows the designer to identify these paths and acts to assign page labels to buttons to minimize the difficulty of executing these button sequences. MFDTool allows the designer to specify as many of these paths as desired.

5. **Pages to far buttons:** Sometimes a designer may want a set of labels to be separated as much as possible. This could be useful to insure that a user does not confuse similar labels that are functionally different. It could also help the designer organize labels into different positions on the screen. Use of this constraint likely corresponds to guideline (10), above.

6. **Parent to child variability:** This constraint allows a designer to impose certain types of consistency on the layout design. This constraint attempts to minimize, among specified pages, the variability (standard deviation) in interaction coefficients between the buttons assigned to a page's parent and to itself. For example, if Euclidean distance is used, then specified pages will be assigned to buttons so that clicking on the button of a page's parent will then require a move of a constant distance to reach the desired page's button. This would likely provide a consistent "motor memory" that might make learning and using the MFD easier.

In MFDTool, each constraint has a corresponding numerical cost function that measures how well a constraint is being satisfied by the current MFD design. Smaller cost values correspond to better designs relative to that constraint. An optimization algorithm searches through a variety of MFD designs to find one that minimizes (or nearly so) the sum of costs.

The next two sections are fairly complex, involving a mathematical description of interactions and costs, and they can be skimmed (but not skipped) by beginners who want to learn how to use the methods of MFDTool, but are not yet interested in the underlying principles.

2.3 Interactions

MFDTool optimizes the assignment of page labels to buttons. The optimization is relative to specified cost functions. Each cost function must be defined relative to a designer-specified interaction. Each interaction is defined by a set of *interaction coefficients* that identify the relationships every pair of buttons. For example, a Euclidean distance interaction would consist of interaction coefficients that identify the Euclidean distance between each pair of buttons. As another example, some MFDs have the user push arrow keys to "tab" through buttons. The interaction coefficients here might indicate the number of pushes that are required to go from one button to another.

Interaction coefficients need not be true distances in the mathematical sense. For example, in some situations it may be easy to go from the left side of the MFD to the right side, but difficult to go the other way. The interaction coefficients that would define such a relationship would not be symmetrical, and hence would not correspond to a true distance. Nevertheless, a designer can generally think of the interaction coefficients as identifying how difficult it is for a user to go from one button to another. Larger numbers indicate more difficulty. The task of MFDTool is to assign page labels to buttons to minimize cost functions relative to the interaction coefficients.

In general, a designer can specify the interaction coefficients to be whatever is desired. It may be a difficult task in some instances to specify the interaction coefficients for a particular

MFD. It requires an understanding of the human-computer interface, and perhaps a model of how the user will interact with the interface. In principal, this type of analysis should be carried out for any human-computer interaction design. MFDTool does provide a means of calculating five sets of interaction coefficients that correspond to some types of human-computer interactions. These are described below.

2.3.1 Fitt's movement

A common form of interaction with an MFD is finger-pointing, where a user presses buttons by moving his finger (or perhaps a mouse or pointer pen) from one button to another. There is a well established model of how long it takes people to move a pointer over a given distance to a target of a given size. MFDTool uses a form of Fitt's Law that says that the movement time, $I[i,j]$ between button i and button j is:

$$I[i,j] = I_m \log_2 \left(\frac{2H_{ij}}{S_j} + 1 \right). \tag{1}$$

Here, H_{ij} is the center-to-center Euclidean distance between the buttons; S_j is the size of the second button (defined as the minimum of button height and width), $\log_2$ is the logarithm in base 2, and I_m is a parameter with units milliseconds/bit. I_m is empirically measured, and, for finger movements, values between 70 and 120 ms/bit are common. MFDTool uses $I_m = 100$ ms/bit.

In the context of moving through an MFD hierarchy of information, Fitt's Law is almost surely a lower limit of execution time, as a user may need to read labels to remember which button to press next.

2.3.2 Euclidean distance

This set of interaction coefficients describes the physical distance between button centers. The interaction coefficient for two buttons i and j, centered on (x_i, y_i) and (x_j, y_j), respectively, is computed as

$$I[i,j] = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

2.3.3 City-block distance

This set of interaction coefficients describes the city-block distance between button centers. The interaction coefficient for two buttons i and j centered on (x_i, y_i) and (x_j, y_j), respectively, is computed as

$$I[i,j] = |x_i - x_j| + |y_i - y_j|,$$

where $|\ |$ indicates absolute value.

2.3.4 X-directed

This set of interaction coefficients describes the horizontal distance and direction of button centers. The interaction coefficient for two buttons i and j centered on (x_i, y_i) and (x_j, y_j),

respectively, is computed as

$$I[i,j] = x_j - x_i.$$

Position $(0,0)$ corresponds to the upper left corner of the MFD screen. A positive interaction coefficient indicates that the second button is to the right of the first. A negative coefficient indicates that the second button is to the left of the first.

This coefficient does not really indicate difficulty of movement between a pair of buttons, but the absolute value of this coefficient would indicate the horizontal distance between buttons.

2.3.5 *Y*-directed

This set of interaction coefficients describes the vertical distance and direction of button centers. The interaction coefficient for two buttons i and j centered on (x_i, y_i) and (x_j, y_j), respectively, is computed as

$$I[i,j] = y_i - y_j.$$

Position $(0,0)$ corresponds to the upper left corner of the MFD screen. A positive interaction coefficient indicates that the second button is above the first. A negative coefficient indicates that the second button is below the first.

This coefficient does not really indicate difficulty of movement between a pair of buttons, but the absolute value of this coefficient would indicate the vertical distance between buttons.

2.4 Constraint costs

For each interaction, the designer can associate any number of constraints. MFDTool acts to minimize the cost functions associated with these constraints. This section mathematically defines the cost functions.

2.4.1 Global difficulty

When a sequence of button presses, $b_1, b_2, ..., b_m$, is to be executed during access of information, MFDTool adds up the interaction coefficients related to moving from the first button to the second, the second button to the third, and so on. Thus, the total difficulty executing a sequence of button presses will be:

$$D = \sum_{i=1}^{m-1} I[b_i, b_{i+1}], \tag{2}$$

where there are m button presses in the sequence, and $I[b_i, b_{i+1}]$ is the interaction coefficient between successive buttons.

Once the interaction model is defined by specification of all the $I[b_i, b_{i+1}]$ terms, MFDTool can calculate the total difficulty involved in reaching a desired information label by looking at the sequence of button pushes necessary to reach that page label from the top level of

the MFD hierarchy. The cost function for global difficulty is the average difficulty needed to reach any MFD page label, for a given interaction:

$$C_1 = \sum_{j=1}^{n} D_j p_j, \tag{3}$$

where n is the number of information labels in the hierarchy, D_j is the total difficulty involved in executing the sequence of button presses to reach page label j, and p_j is the proportion of time that page label j is needed by the user. As the assignment of page labels to buttons is modified, the value of D_j changes. MFDTool tries to assign page labels to button presses so that labels with larger p_j values have smaller D_j values, thereby minimizing average difficulty.

2.4.2 Pages to close buttons

To help users learn and remember where information is located on an MFD screen, designers often group sets of labels to be on nearby locations. A commonly used technique is to place labels that are related to each other on nearby buttons. A designer may, for example, want to create ordered lists of items on a single MFD screen and may also want to insure that related labels on different screens are associated with nearby buttons.

MFDTool defines "closeness" relative to the interaction coefficients between buttons. Thus, if a designer constrains page labels $L_1, ..., L_k$ to be as close as possible, and they are currently assigned to buttons $b(L_1), ..., b(L_k)$, then the quantitative cost of these assignments is:

$$C_2 = \sum_{i=1}^{k} \sum_{j=1}^{k} I[b(L_i), b(L_j)]. \tag{4}$$

Here $I[b(L_i), b(L_j)]$ is the interaction coefficient from button $b(L_i)$ to button $b(L_j)$. If the interaction defines a distance, then C_2 equals zero when the labels in this constraint are all on different pages and are associated with the same button. If some of the labels are on the same screen, then C_2 will likely have a nonzero minimum, as two labels cannot simultaneously be associated with the same button on the same screen. Whichever the case, MFDTool tries to assign information labels to buttons in a way that minimizes C_2.

2.4.3 Pages to fixed buttons

Sometimes a designer may want to constrain some page labels to a particular button or set of buttons. This could occur for example, if a designer, to stay consistent with other displays, wants an EXIT label always placed on the lower left button. Or, a designer may want geographical topics to have corresponding positions on the MFD screen (e.g., left to left). All of these constraints can be imposed in MFDTool.

This constraint cost measures how close the page labels are to their restricted buttons. As with the other costs, MFDTool defines "closeness" relative to the interaction coefficients between buttons. Thus, if the designer constrains page labels $L_1, ..., L_k$ to be restricted to

buttons $b_1, ..., b_h$, and each label is currently assigned to buttons $b(L_1), ..., b(L_k)$, then the quantitative cost of these assignments is:

$$C_3 = \sum_{i=1}^{k} \min_{j=1,...,h} I[b_j, b(L_i)]. \quad (5)$$

The term inside the summation compares the currently assigned button for label L_i with each of the allowable buttons and takes the minimum interaction coefficient. Thus, if the interaction defines a distance, then when all labels are assigned to one of the allowable buttons the minimum interaction coefficients are zero and the total cost is zero. When a constrained label is not assigned to an allowable button, the cost is incremented by the minimum interaction coefficient between the assigned button and one of the allowable buttons.

2.4.4 Path difficulty

C_1, above, measures the average difficulty required to search for a page label, starting from the top of the hierarchy and taking the most direct route to that label. However, depending on the MFD, not all searches are of that type. It is frequently the case that a user needs to gather a number of different types of information from different screens in the MFD. The designer may include shortcuts or hyperlinks that allow the user to quickly travel along such paths of pages. Cost C_1 cannot account for these types of situations because the use of shortcuts means that there are multiple (usually infinitely many) ways to reach a label. For these types of situations, the designer must specify the sequence, or path, of pages the user goes through to perform a required task. Once this path is specified, MFDTool acts to minimize the interaction coefficients along that path by associating page labels to buttons, much as for cost C_1.

The quantitative definition of cost is much as for C_1, except the designer must identify the path of page labels that the user steps through (for C_1 the computer could do this because each page label has a unique position in the hierarchy). The designer identifies an ordered sequence of page labels $L_1, ..., L_k$ for which total difficulty is to be minimized. If each page label is currently assigned to buttons $b(L_1), ..., b(L_k)$, then the quantitative cost of these assignments is:

$$C_4 = \sum_{i=1}^{k-1} I[b(L_i), b(L_{i+1})]. \quad (6)$$

MFDTool tries to minimize this cost through the assignment of page labels to buttons.

In some MFD applications, minimization of difficulty along these paths may be the most important job for the designer. By their very nature, such sequences must be specified by the designer.

2.4.5 Pages to far buttons

To help users learn and remember where information is located on an MFD screen, designers often group sets of labels to be at distinctive locations. A commonly used technique is to place labels that are unrelated to each other on buttons that are separated on the display.

MFDTool defines "farness" relative to the interaction coefficients between buttons. Thus, if a designer constrains page labels $L_1, ..., L_k$ to be as far apart as possible, and each label is currently assigned to buttons $b(L_1), ..., b(L_k)$, then the quantitative cost of these assignments is:

$$C_5 = \sum_{i=1}^{k} \sum_{j=1}^{k} (I_{\max} - I[b(L_i), b(L_j)]) . \tag{7}$$

Here $I_{\max}$ is the largest possible interaction coefficient between any buttons on the display and $I[b(L_i), b(L_j)]$ is the interaction coefficient from button $b(L_i)$ to button $b(L_j)$. This constraint is most useful for an interaction that defines a distance, so that C_5 equals zero when the labels in this constraint are as far apart as possible. It may not be possible to drive C_5 to zero when there is a larger number of constrained pages than can be placed on the farthest apart button pairs.

2.4.6 Parent to child variability

Guideline (8) suggests that related functions on separate pages should be placed on similar locations. More generally, this guideline suggests that the position of functions should be *consistent* across the MFD design. However, the guideline only specifically mentions positional consistency. There are other types of consistency that a designer might also want to consider.

If the MFD interaction requires movement of the arm or hand, there may be motoric memory of the motions required to access information. For these types of interactions, there may be an advantage for a design that provides consistency of movements, rather than consistency of locations. For example, perhaps a `Cancel` button should always be located opposite the just selected button. This would make the `Cancel` button assigned to different buttons on different screens, but in a consistent relative location across screens. If a designer decides that this type of consistency is important for their system, then this constraint can be used to describe a cost for violating a specified consistency.

If a designer constrains page labels $L_1, ..., L_k$, each with a hierarchical parent page $P_1, ..., P_k$, to be in a consistent relative location as the user goes from the P_i to L_i, and each label is currently assigned to buttons $b(L_1), ..., b(L_k)$, then the quantitative cost of these assignments is:

$$C_6 = \sqrt{\frac{\sum_{i=1}^{k} I[b(P_i), b(L_i)]^2 - (\sum_{i=1}^{k} I[b(P_i), b(L_i)])^2/k}{k}}. \tag{8}$$

This equation is basically a calculation of standard deviation. $I[b(P_i), b(L_i)]$ is the interaction coefficient from the button of the hierarchical parent of label L_i, $b(P_i)$, to button $b(L_i)$. The standard deviation of these interaction coefficients is the cost for this constraint. If used with the X-directed and Y-directed interactions, this constraint allows for direction-specific consistency across page label locations. If all the interaction coefficients of constrained pages are the same, the cost is zero.

2.5 Weighting costs

The designer needs to identify the relative importance of different constraints so that MFDTool produces the desired result. It is common for constraints to be in conflict with each other. In anticipation of such conflicts the designer needs to indicate a weight, λ, for each constraint cost. For example, if the designer wants to be certain that the `EXIT` label is always on the lower left button, even if such assignment means an increase in other costs, then the weight for the `EXIT` constraint might be set larger than the weights for other costs.

MFDTool tries to minimize the weighted sum of constraint costs:

$$C = \sum_{i=1}^{n} \lambda_i R_i. \tag{9}$$

Here, there are n constraints defined by the user, and R_i corresponds to the cost associated with constraint i.

There is no way for MFDTool to advise the designer on how to set the weights. The default is the value one, but it is merely a starting point and not intended as a reasonable choice. The values of the weights have a great effect on the resulting MFD design, and it is not unusual for a designer to tweak the weights to insure that one constraint is satisfied over another. The use of extremely large weights, relative to others, is often not effective because it sometimes hinders the optimization process (next section). In general, the designer should set the weight on a constraint just high enough to insure that the constraint is satisfied. This often involves trial and error.

2.6 Optimization

Once a total cost function is defined, one can use any number of algorithms to find the assignment of page labels to buttons that minimizes that cost function. MFDTool currently includes two optimization techniques: gradient descent and the simulated annealing algorithm.

The gradient descent algorithm is a variation of hill-climbing algorithms. In a hill-climbing (or hill-descending, only the sign needs to be changed) algorithm, the system is initialized to a particular state (e.g., mapping of labels to buttons) and the cost is calculated for that state. One of the variables of the problem (e.g., a label) is randomly selected and modified (e.g., moved to a new button). A new cost value is calculated, and if the new cost is less than the old cost, the change is kept, otherwise the change is undone. In this way, the system converges to a state where any change would lead to an increase in cost (e.g., where any change in the mapping would be worse). Hill-climbing techniques have a tendency to get stuck in local minima of cost because they never accept changes that increase cost. In complex problems, hill-climbing methods can easily get trapped in a state where any change only increases cost but the global minimum is very different, with a much smaller cost. To avoid this problem, MFDTool proceeds through many runs of the optimization with different random initial designs, and keeps track of the overall best design. The success of this approach depends on the complexity of the cost-space of the system (which is generally unknown). In practice, this algorithm is fast and often ends up with a good optimization.

Simulated annealing is another variation of hill-climbing algorithms that addresses the local minima problem by introducing a controlled way to climb out of local minima and end in a state with the global minimum of cost. By analogy, one would probably, at some point during a hike, need to go down a ravine or a small slope to climb to the top of a mountain. Simulated annealing is a stochastic algorithm that at first accepts changes even if they lead to larger costs. As time progresses a temperature parameter gradually decreases (this is the annealing) so that it becomes less likely that a change leading to an increase in cost is accepted. As the temperature becomes small the algorithm becomes essentially hill-climbing. As long as the temperature decreases slowly enough and enough changes are considered at each temperature level, simulated annealing is statistically guaranteed to find the global minimum of a problem. In practice, though, the necessary temperature schedule is too slow and the number of changes at each level is too large, so simpler approaches are taken that are faster, but less certain to find the global minimum.

In simulated annealing, the initial temperature, T, is set large enough that many state changes are accepted even if they lead to a cost increase. Changes are made by randomly selecting an MFD page label. Its button assignment is noted, and a new button assignment is randomly selected. The selected label swaps positions with whatever (perhaps nothing) is at the new button assignment. After each change, new cost, C_{new}, is calculated and compared to the cost before the change, C_{old}. The change in cost, $\Delta C = C_{\text{new}} - C_{\text{old}}$, is calculated. If the cost change is negative, the change is kept. If the cost change is positive, the change is kept when a random number between zero and one is greater than

$$p = \exp(-\Delta C/T). \tag{10}$$

This relationship means that when ΔC is much smaller than T, p is close to one, and lots of changes are kept. As T gets smaller than ΔC, p gets closer to zero, and changes are not kept very often. Statistically then, the system is more likely to be in a state with a low cost. As T decreases, the system tends to be stuck in a state with very low cost.

To insure that the statistical situation is close to reality, one needs to implement many changes at every temperature level. After these changes, the temperature is multiplied by a number between zero and one. The process is then repeated for the new temperature. The whole process stops when it seems that the temperature is so small that the system is trapped in a particular state (as in hill-climbing). MFDTool reaches this conclusion when several changes in temperature have not produced any changes in cost.

At the end of the simulated annealing process, the system should be in a state with a low (but perhaps not optimal) cost. Being certain of finding the true optimal state with the absolute lowest possible cost would be prohibitively difficult and would likely require a supercomputer, even for relatively small MFDs.

In theory, simulated annealing should give a better optimized result than the gradient descent algorithm. However, in practice simulated annealing's performance depends on the parameters selected, so in many instances the gradient descent procedure is preferable.

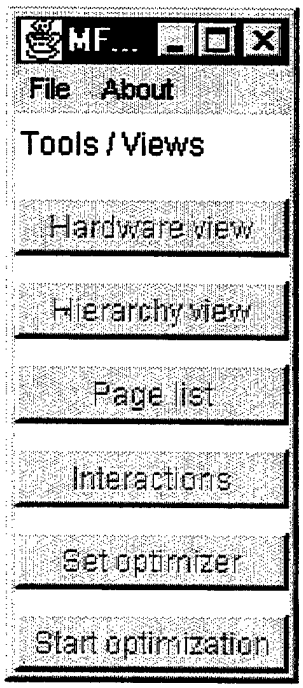


Figure 1: The MFDTool startup window, as seen in Windows 98.

3 Using MFDTool

MFDTool needs three basic types of information: a description of the MFD hardware components, a description of the hierarchical arrangement of information, and designer-specified interactions and constraints that allow MFDTool to decide how the information should be mapped to the hardware components. In earlier versions of MFDTool, each MFD needed to be completely designed from scratch. MFDTool 1.2 introduced the capability to modify the hardware, hierarchy, interaction, and constraint information so that it may be easier to copy and edit a previously created MFD (a sample is included). To avoid confusion, in MFDTool 1.3, this is the only way to create to MFDs.

Start MFDTool in the manner appropriate for your operating system (see section 1 on installation). A window should appear like that in Figure 1, although it will vary somewhat from one operating system to another.

3.1 Opening a previously saved MFD

From the MFDTool start window select the `File` menu, and select `Open`. An open file window should appear. Go to the directory `Examples` in the MFDTool directory and open the file `ATM.mfd`. The buttons on the `MFDTool` window will become clickable.

3.2 Saving an opened MFD

From the MFDTool start window select the `File` menu, and select `Save`. A save file window should appear. Go to the desired directory and save the file. It is recommended that you use the extension `.mfd`, but this is not technically necessary.

4 MFD hardware

MFD hardware corresponds to the size of the MFD screen, the number of buttons, the size of each button, and the location of each button on the MFD screen. MFD hardware information can be specified at the creation of a new MFD, and can be edited on an open MFD. An `MFD Hardware` window is designed to give an idea of the current assignment of information labels to buttons. This allows the designer to determine whether the assignment is consistent with what was intended. This window provides mechanisms for editing hardware information and also interacts with other tools in MFDTool.

From the MFD Start window, click on the button labeled `Hardware view`. (An MFD must be opened for this button to be clicked.) Figure 2 shows what the window titled `MFD Hardware: ATM` should look like on Windows 98, if the MFD `ATM.mfd` was opened.

Notice that the two top left buttons have text (`OK Password` and `Cancel`) on them. This text corresponds to page labels that are currently associated with these buttons. Not all of the text may be visible on a button, but this should be of no concern. MFDTool does not try to exactly emulate what the MFD hardware will look like. Many devices place the text to one side of buttons, while others place the text directly on the buttons. MFDTool does not distinguish between these display types. It only cares about the association between the label and the button. Likewise, MFDTool does not consider whether label text will fit on a button or next to a button. The designer must handled these types of concerns by choosing appropriate text and font sizes. More generally, the actual text of the labels is for the benefit of the designer (and ultimately the user), but MFDTool makes no use of the text itself.

Clicking on a button with a label reconfigures the `MFD hardware` window to show the contents of the page associated with that label. After selecting `OK Password`, the `MFD Hardware` window will look like the screenshot in Figure 3, with new labels on the buttons. The click "moved" the display screen down one level in the hierarchy to show the contents of the `OK Password` page. You can continue to select labels to move further down the hierarchy. However, you cannot yet move back up the hierarchy because any necessary hyperlinks that would move up the hierarchy have not been specified (below).

4.1 Editing MFD hardware

Once an MFD is loaded into MFD tool, a designer can edit hardware information through the pull-down menus on the `MFD Hardware` window. Figure 4 shows the `MFD Set Frame Info` window that appears upon selection of the `Edit frame` menu. In this window the designer can set the size of the MFD frame, and can also modify the name of the MFD (this name appears on the title of each MFDTool window).

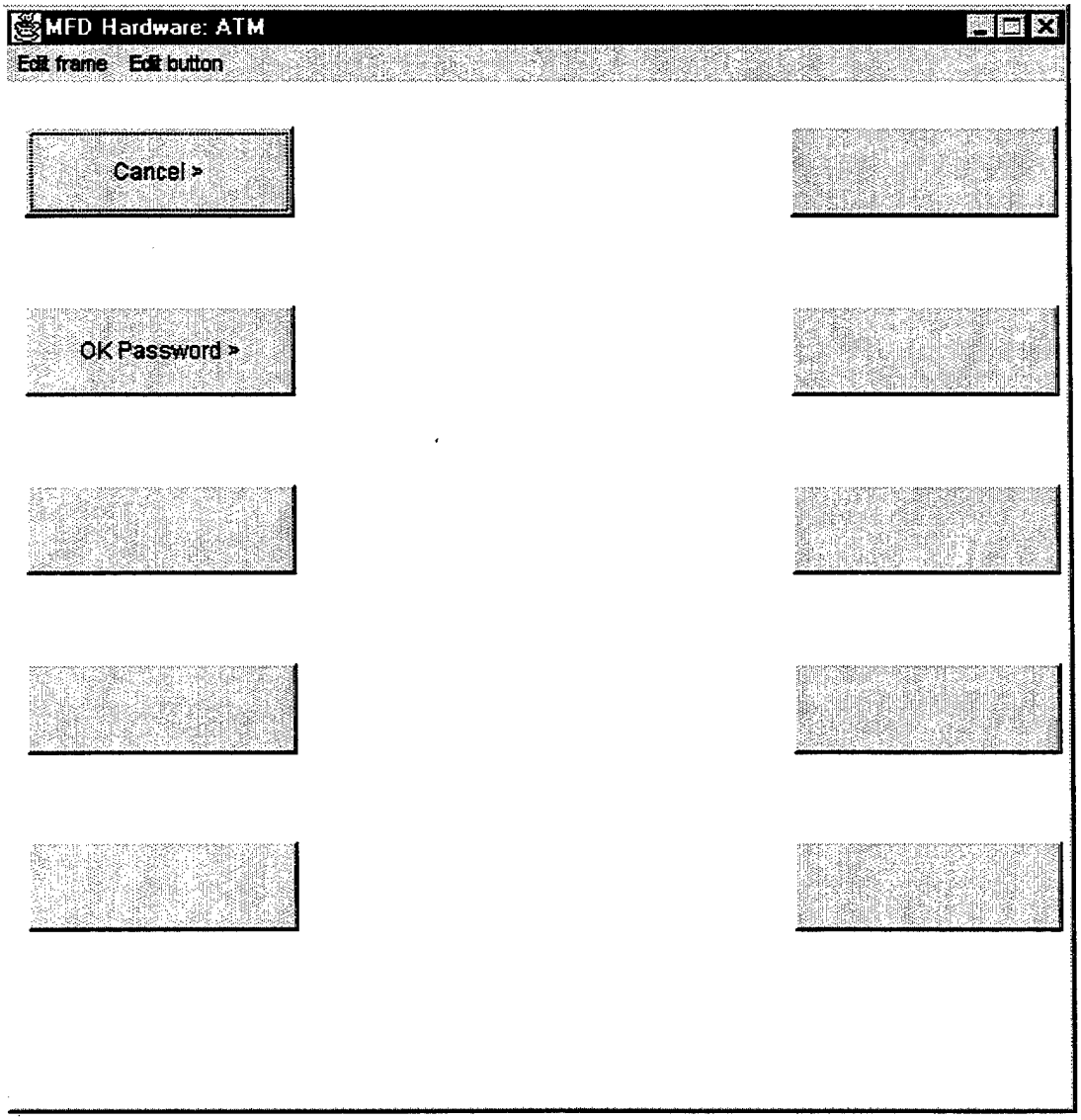


Figure 2: The MFD Hardware window shows the physical layout of the MFD and its buttons.

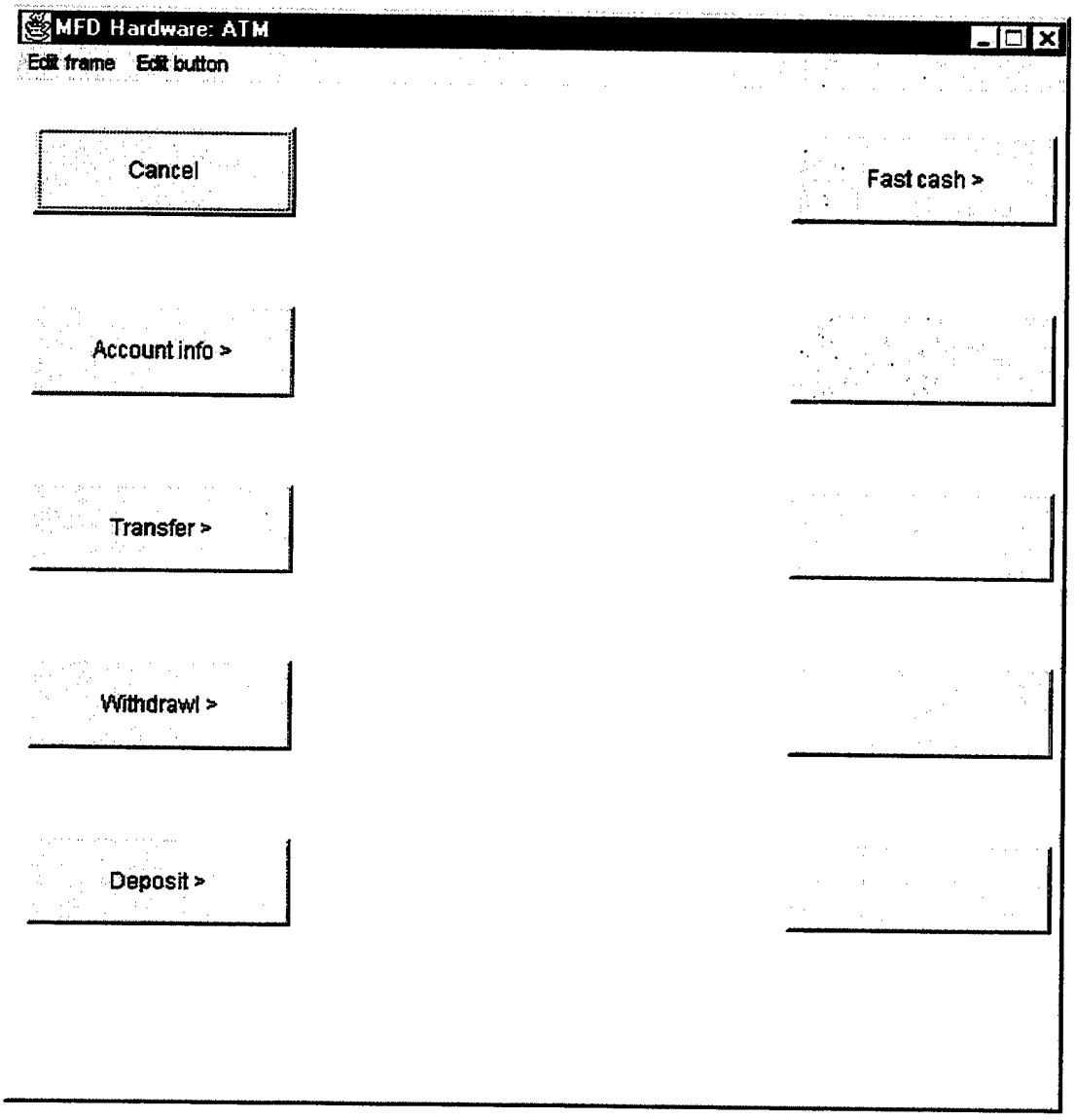


Figure 3: The MFD Hardware window after selecting the `OK Password` label shows the labels on the selected page.

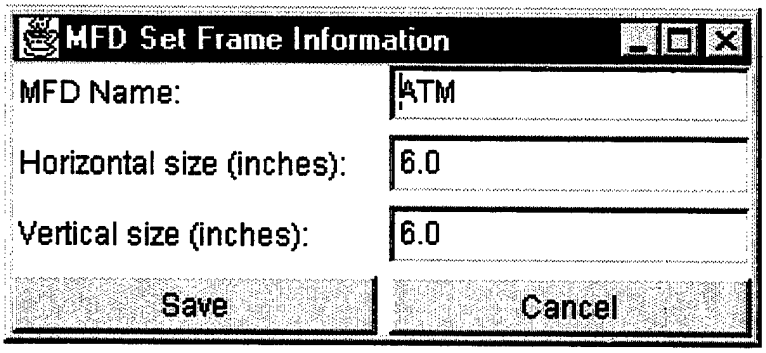


Figure 4: The `MFD Set Frame Information` window allows the designer to modify the name of the MFD and set the size of the MFD frame.

Physical attributes of buttons can be edited in two ways. The easiest is to press the control key and click on a button. A new window will appear, as in Figure 5, for the button on the top-left. From this window, the designer can provide a more informative name for a button (e.g., *Upper Left* would be more informative than *b0*). The designer can also modify the size and location of the button.

The same window can be reached through the `Edit button` pull-down menu from the `MFD Hardware` window. This menu contains an item for each button (in this case named `b0,....b9`). Selecting the menu item `b0` causes the pop-up of the window show in Figure 5. Use of the menu is most useful for situations where a button is accidentally placed off the MFD screen, and thereby cannot be clicked for further editing.

A special pair of selections in the `MFD Set Button` frame is the Fitts sizes. The Fitts sizes allow a designer to specify virtual sizes that should be incorporated in the calculation of Fitt's Law. This is useful when the sides of the MFD restrict movement beyond the edge of the screen. In such a case a user can make a ballistic movement toward a corner and know that he will stop moving at the corner. If the button is directly on that corner, then very fast movements can be made to corner buttons. This can be emulated in the calculations of Fitt's Law by giving these corner buttons very large sizes. If none of these issues concern you, just leave these numbers alone, their default sizes are the same as the physical size of the buttons.

Entering information in the `MFD Set Button Information` window and clicking on the `New` button creates a new button with the specified information. Clicking on the `Delete` button deletes the selected button (whether any information has been changed or not). There is no undelete feature, so use the delete command carefully.

Any changes made to the MFD hardware are automatically updated in other components of MFDTool. In some cases changes in MFD hardware may require additional effort from the designer to update some constraints (e.g., that involved a now deleted button) or interactions (e.g., to define new interaction coefficients). By default, deleted buttons are removed from any constraints that involved them, interaction coefficients are removed for a deleted button, and interaction coefficients for a new button are set to zero. Changing the size or location of a button does not force an update of its interaction coefficients, which must be done by

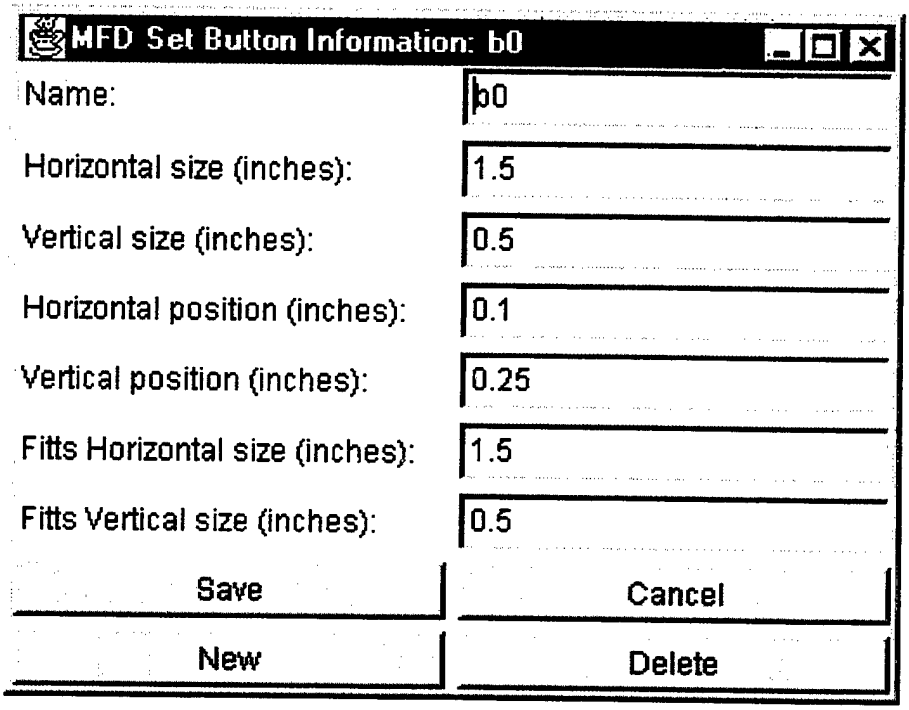


Figure 5: The `MFD Set Button Information` window allows the designer to modify button properties, delete a button, and create a new button.

the designer.

It is possible that the size and placement of the hardware frame and buttons are slightly different than specified. However, the information used by MFDTool in any computations is exactly as specified. Sometimes Java windows and buttons are created slightly different than as specified. The calculations of distance and sizes (which are involved in the MFDTool defined interfaces) will be based on the designer-supplied numbers and not on how the `MFD Hardware` window looks.

5 MFD hierarchy

An MFD hierarchy corresponds to labels and their hierarchical arrangement. Information on the individual pages of the hierarchy can be viewed in an `MFD Page List` window, while movement up an down the hierarchy is facilitated by an `MFD Hierarchy` window.

5.1 Views of MFD hierarchy

MFDTool distinguishes between three types of MFD pages. A *parent* is a page such that when that page's label is selected the MFD screen changes to reveal the contents of the page, which contains new page labels that can also be selected. A *terminator* is a page that, when its label is selected, the MFD screen may reveal additional information, but does not show new labels for new pages that can also be selected. A *hyperlink* is a page that, when its label is selected, the MFD jumps to another page someplace else in the hierarchy. Hyperlinks allow the designer to provide shortcuts between distant MFD pages. Hyperlinks also can provide a means to move up the MFD hierarchy structure, when required.

Each page has a name associated with it, and that name is the text of the label that, when selected, shows the contents of the page. For a terminator, selecting the button with its label has little effect because MFDTool is not concerned with the content shown on a page, only with the ordering of pages that might be "children" of a parent page.

The MFD information is displayed in multiple ways. First, as already noted above (Figures 2 and 3), the `MFD Hardware` window shows the MFD information, with each page name as text on its currently associated button. Second, an `MFD Hierarchy` window provides a listing of the contents of the currently selected page. Figure 6 shows what this type of window looks like after loading the file `ATM.mfd` as above. The names associated with parent pages are given an additional ">" at their end, so the designer can quickly recognize that selecting such a page will produce movement in the hierarchy.

Movement through the hierarchy of information is done by selecting a text row. Such a selection has several effects. First, the `MFD Hierarchy` window changes to show the contents of the selected page (Figure 7). Second, the `MFD Hardware` window also changes. The two windows are (almost) yoked so that changes to one results in changes to the other. There is an exception to this yoking for terminators and hyperlinks, as described below. Third, a status line at the bottom of the `MFD Hierarchy` window shows the path of pages leading to the currently selected page. This helps the designer keep track of which page is currently selected in the hierarchy. From the `MFD Hierarchy` window, the designer can move up the

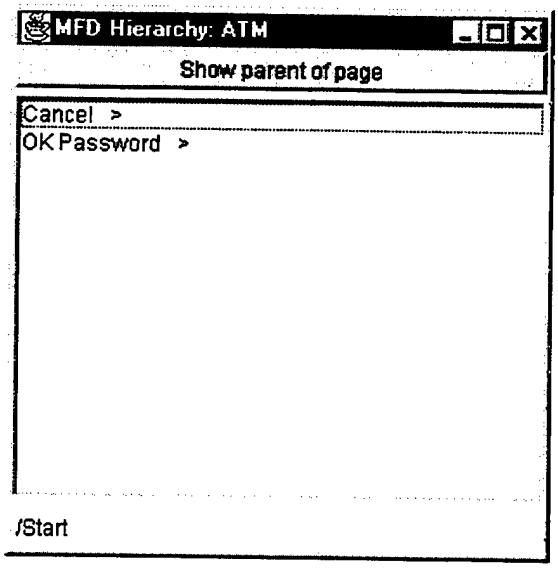


Figure 6: The `MFD Hierarchy` window as it appears after opening the file `ATM.mfd`.

hierarchy by selecting the `Show parent of page` button, which automatically moves up to the parent of the currently selected page.

The yoke between the `MFD Hardware` and `MFD Hierarchy` windows is not absolute. For example, selecting a terminator page in the `MFD Hierarchy` window makes that page the currently selected MFD page and allows the designer to impose constraints on that page (described below). On the `MFD Hardware` window, there is no noticeable change in the MFD buttons, because MFDTool does not show contents of a terminator page but instead shows the contents of the page's parent. Also, selecting a hyperlink in the `MFD Hierarchy` window does not automatically cause the `MFD Hardware` window to jump to the hyperlink target. Once selected in the `MFD Hierarchy` window, clicking on the name of the hyperlink a second time will cause the `MFD Hardware` window to jump to the hyperlink target.

The selection of pages from the `MFD Hardware` or `MFD Hierarchy` windows is also yoked to a third display of MFD page names in the `MFD Page list` window. Figure 8 shows this window after the file `ATM.mfd` is loaded. All forty-two pages in the ATM hierarchy are listed on the bottom of the `MFD Page list` window. Each page has its name and its type: Parent (P), Terminator (T) or Hyperlink (H). In addition, a hyperlink page has an added "-" to indicate that it needs to have its hyperlink target set by the designer. Selecting a page from this list updates the `MFD Hardware` and `MFD Hierarchy` windows to display that page (or that page's parent, if the selected page is a terminator or a hyperlink). Each page listed also has a number, $p = 0.023809524$, which indicates the default proportion of times that this page will be needed by the user. These proportions are relative to the currently selected interaction (describe below). MFDTool's default is to assume that every page is needed equally often, so this number is one over the number of pages in the hierarchy, 1/42. These proportions can be set by the designer, as described below.

The yellow area at the top of the `MFD Page list` window contains information about the

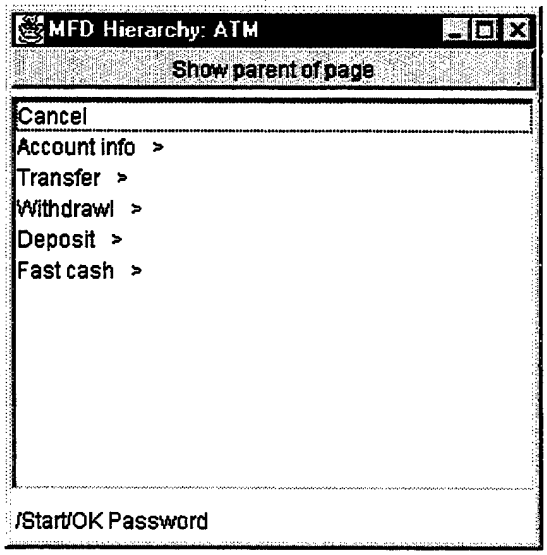


Figure 7: The `MFD Hierarchy` window as it appears after selecting the page name `OK Password`.

currently selected page. In Figure 8, the selected page is the `Start` page, which corresponds to the very top of the MFD hierarchy. From left to right in the `MFD Page list` window, one finds the following information. This label is associated with button `b0`, as indicated in the list of "Button assignment." The name of the label is "Start". Its proportion is the default 0.023809524; and the checkboxes to the right of the proportion text field indicate that this proportion is "free". This indicates that as other pages change their proportion to be a "fixed" amount set by the designer, the proportion for this page will adjust accordingly. The page type is "Parent". The parent of this page is itself; which is only going to be the case for the very top page of the hierarchy. No hyperlink is required because this page is not a hyperlink. Figure 9 shows that the information is updated when the `OK Password` page is selected.

5.2 Editing an MFD hierarchy

MFDTool provides a number of tools to modify, add, delete, move, and duplicate MFD pages. All of these capabilities are provided on the `MFD Page list` window.

5.2.1 Setting hyperlinks

A hyperlink page contains a shortcut to a different part of the MFD hierarchy. Selecting a properly defined hyperlink in the `MFD Hardware` window makes the hyperlink target page the currently selected page and displays that page (or its parent if the target is a terminator). When the directory structure that defines the MFD hierarchy is loaded into MFDTool, there is no way to identify the targets of hyperlink pages. The designer must define the hyperlinks within MFDTool. This is done from the `MFD Page list` window.

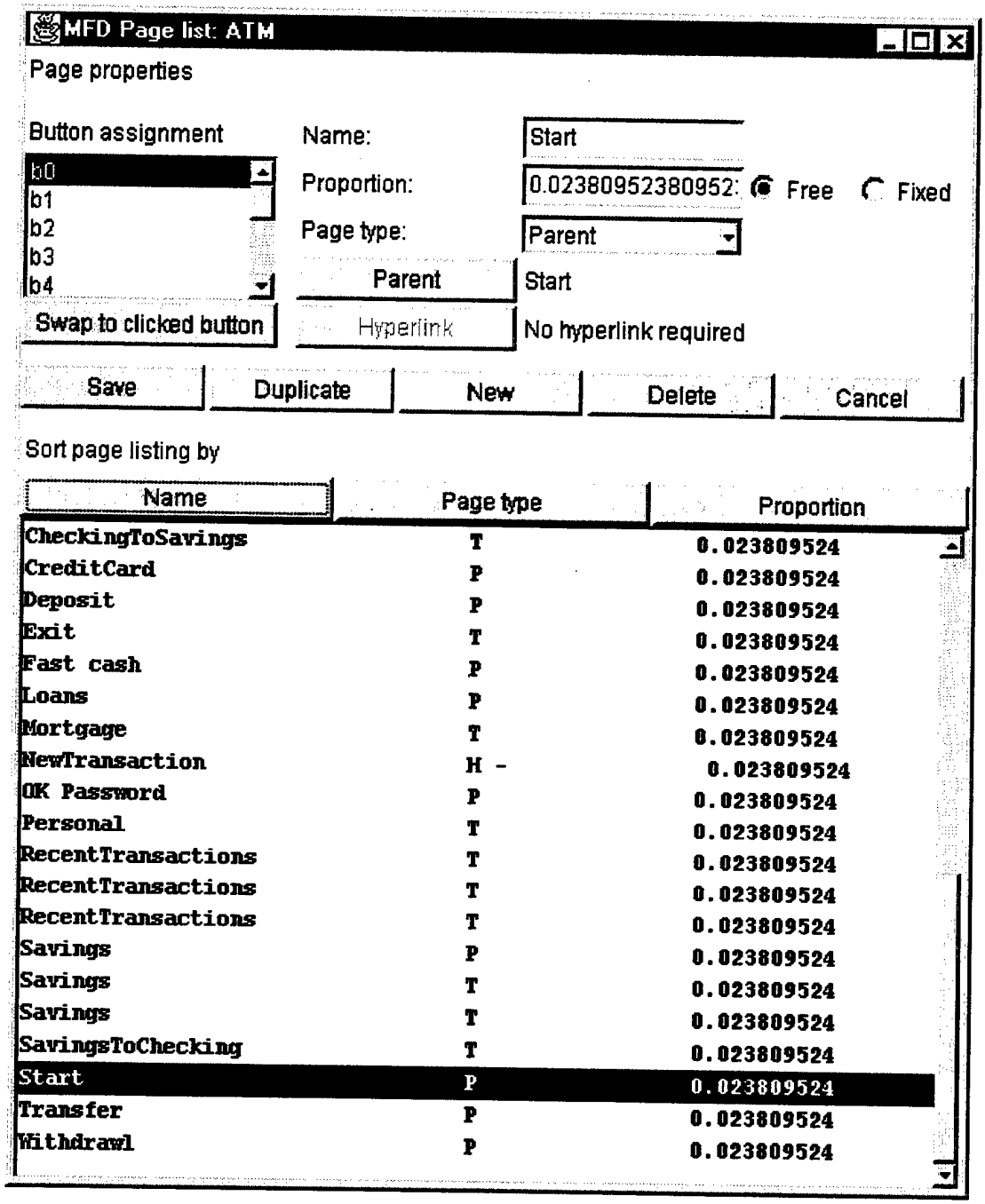


Figure 8: The `MFD Page list` window as it appears after opening the file `ATM.mfd`.

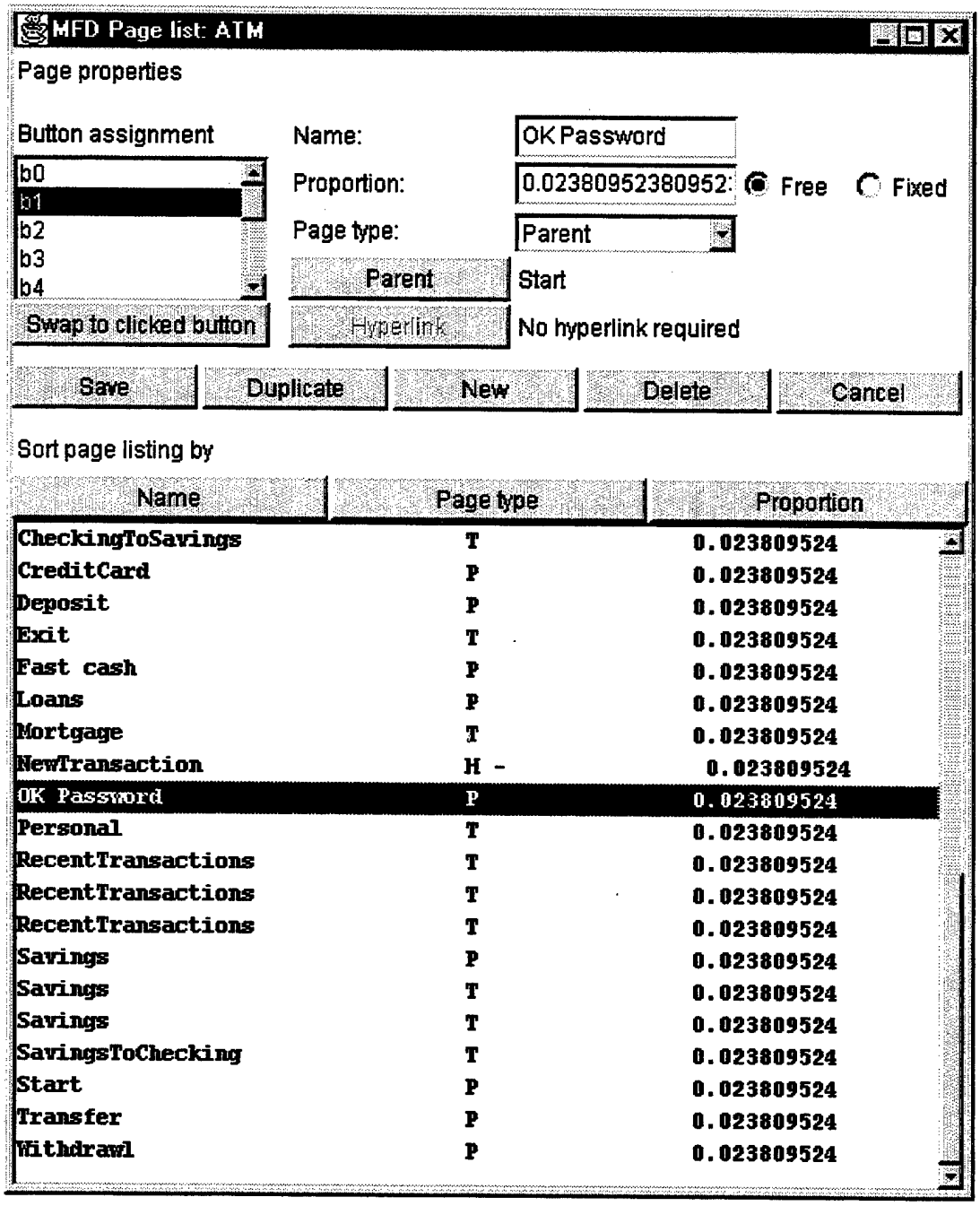


Figure 9: The `MFD Page list` window as it appears after selecting the page name `OK Password`.

Hyperlinks that have not had their target defined are shown in the page listing on the MFDTool window with type "H-". If a hyperlink is selected from this list (or from the MFD Hierarchy window), the top part of the MFD Page list window displays information on the page. From the MFD Page list window select the first Cancel hyperlink page. The window should look like in Figure 10. To set the hyperlink, click on the Hyperlink button. The button will change its label to read Set hyperlink and the text to the right will read "Select a hyperlink target." This enters the MFD Page list screen into a mode where selecting an MFD page (through the MFD Page list, MFD Hardware, or MFD Hierarchy window) makes it a possible hyperlink target for the edited page. In this mode, the name of the selected page appears to the right of the Set hyperlink button. Select the Cancel parent page as the desired hyperlink target, then click on the Set hyperlink button; this defines the currently selected page as the new hyperlink target.

Now click on the Save button, which updates the Cancel hyperlink originally selected. To verify that it has worked properly, open the MFD Hierarchy window and click the Show parent page until you reach the top of the hierarchy. Now open the MFD Hardware window and click on the OK Password button and then the Cancel button. The MFD Hardware window should now display the Cancel page, which contains two labels: Exit and NewTransaction.

You can set the other Cancel hyperlinks in the same way (they should all target the Cancel page). You should also set the hyperlink for page NewTransaction to target the OK Password page. When all this is done, all the hyperlink page types in the MFD Page list will be H instead of H-. Hyperlink targets can be changed at any time.

5.2.2 Modifying page name

Select a page from the MFD Page list, MFD Hardware, or MFD Hierarchy window. The page's name appears in the Name text field. This text field is editable. Simply type in the new name and then click on the Save button. The page label is automatically updated throughout MFDTool (including all constraints, as described below).

5.2.3 Modifying page button assignment

Each page label is assigned to a particular button on the MFD Hardware window. Figuring out the proper assignment is the primary purpose of MFDTool. However, a designer may wish to make the assignment by hand from time to time.

Select a page from the MFD Page list, MFD Hardware, or MFD Hierarchy window. The page's button assignment is highlighted in the "Button assignment" list. For example, if the page $10 is selected, the button titled Left 2 is highlighted. An alternative assignment can be selected in two ways. Either select a different button name from the "Button assignment" list, or click on the Swap to clicked button button and then click on a button on the MFD Hardware window. In either case, then click on the Save button to save the button assignment for the selected page.

If another page label is already associated with the newly selected button, the pages will swap button assignments.

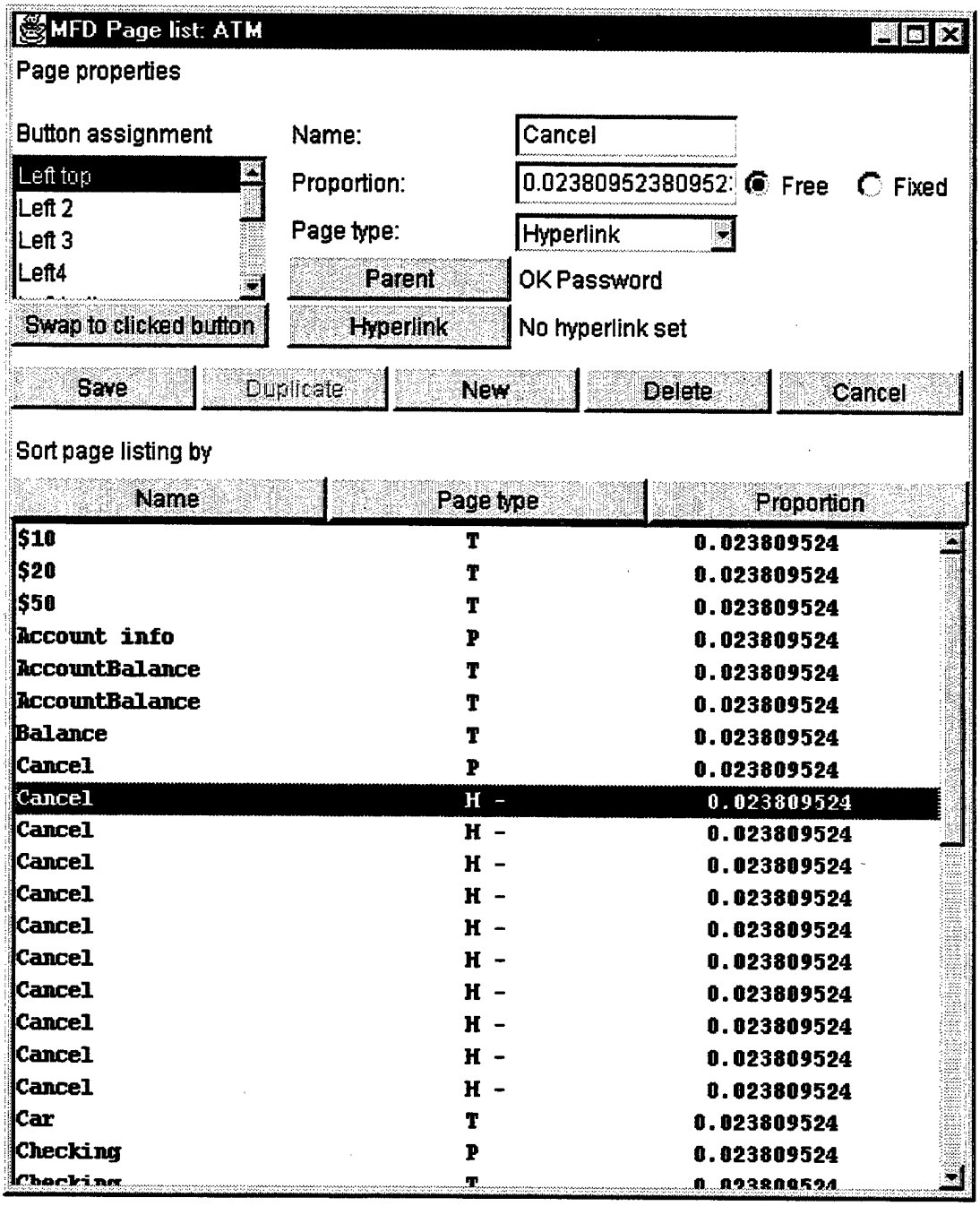


Figure 10: The `MFD Page list` window as it appears after selecting the page `Cancel`.

5.2.4 Moving a page in the hierarchy

The hierarchical arrangement of information is established by the definition of a page's parent. Changing the parent of a page redefines the hierarchical arrangement of information.

Select a page from the `MFD Page list`, `MFD Hardware`, or `MFD Hierarchy` window. The name of the parent of this page is given to the right of the `Parent` button. Clicking on the `Parent` button initiates a mode much like that for the definition of hyperlinks.

For example, select the `CreditCard` page. The name of the parent of this page is `Account info`. Suppose you wanted credit card information to be located on the `OK Password` page, which also lists labels for `Checking`, `Savings`, `Fast cash`, and `Account info`. This is done by setting the parent of the `CreditCard` page. Click on the `Parent` button, it will be relabeled `Set parent` and the text to the right of the button will read "Select a parent page." Click on the `OK Password` label (or choose it from the `MFD Hardware` or `MFD Hierarchy` window). Click on the `Set parent` button. Click on the `Save` button.

You can verify that the `CreditCard` page has moved by looking at the `MFD Hardware` or `MFD Hierarchy` window. Movement of the `CreditCard` page includes movement of any children of that page. The button assignment for a moved page is unchanged, unless another page is already assigned to that button. In the latter case, the moved page is assigned to the first available button. If there are no available buttons under the new parent page, the move will not occur.

5.2.5 Deleting a page

Select a page from the `MFD Page list`, `MFD Hardware`, or `MFD Hierarchy` window. Clicking on the `Delete` button removes that page from the MFD hierarchy. If the page is a parent and there are any children under the page, they are also deleted. The only page that cannot be deleted is the very top page of the MFD hierarchy.

If a deleted page is targeted by a hyperlink page, that hyperlink becomes broken and is indicated with a page type of `H-`. If a deleted page was part of a constraint, the page is automatically removed from that constraint. If the deleted page was part of a `Path difficulty` constraint, the designer will probably want to redefine that constraint.

Be careful with the delete option because there is no undelete option.

5.2.6 Creating a new page

Select a page from the `MFD Page list`, `MFD Hardware`, or `MFD Hierarchy` window. Set whatever properties of a new page you wish (e.g., name, page type, button assignment, parent). Clicking on the `New` button creates a new page that has the specified properties.

If the new page is assigned to a parent that already has as many children as buttons, the new page is not created. If the new page is given a button assignment that is already being used by another page, the new page will be assigned to the first available button.

5.2.7 Duplicating a page

Duplicating a parent page (duplication is not an option for terminators or hyperlinks but they can be easily duplicated with the new page creation) copies not only a selected page but also any other pages that are hierarchically underneath that page. Select a parent page from the `MFD Page list`, `MFD Hardware`, or `MFD Hierarchy` window and then click on the `Duplicate` button. A new page will be created with a name that is the same as the selected page, but with a lead in of "Copy of ". The new page will have the same parent as the originally selected page.

If the parent of the selected page already has as many children as there are buttons, then there will be no room for the newly created page. In such a case the duplication does not occur. If the duplication succeeds, the new page is assigned to the first available button. All the newly created children keep the same properties as their original corresponding pages.

5.2.8 Setting proportions

Proportion indicates how often this page will be accessed by the user, for the currently selected interaction (described below). The default situation assumes that each page will be requested equally often, so with the 42 pages in the MFD, each page has a proportion of 1 over 42, or $p = 0.023809524$. However, typically some pages are used more often than others, and a *Global difficulty* constraint uses these differences to identify how to assign labels to buttons.

For example, the `Start` page is the top of the hierarchy. It is not a page that is ever directly accessed by the user, thus the designer could set its $p = 0$. This is done by selecting the `Start` page and then replacing its proportion with 0.0 in the Proportion text field. Before saving this information, click on the `Fixed` radio button. A `Fixed` proportion is not modified by the proportions of other pages. A `Free` proportion adjusts automatically as other pages change their proportions. After setting the `Start` proportion as fixed, click on the save button. In addition to setting the `Start` page proportion to zero, the proportions of other pages (all free) are changed to $p = 0.024390243$, or 1 over 41. Thus, the sum of all proportions equals 1.0.

Proportions only influence a *Global difficulty* constraint, so there is no reason to define proportions unless this constraint is being used. If multiple interactions are defined and *Global difficulty* is a constraint for each interaction, the proportions will need to be defined separately for each interaction.

6 MFD Interactions

From the `MFDTool` window, click on the `Interactions` button. A window titled `MFD Set Interactions: ATM` will appear. Select the sole item in the list box on the upper left (titled "Initial interaction") and the window will look like that in Figure 11.

The text box at the bottom of the window provides a matrix description of the interaction's coefficients. Each interaction coefficient is a real number that defines a relationship (often distance of some sort) between a pair of buttons. The top row and left-most column

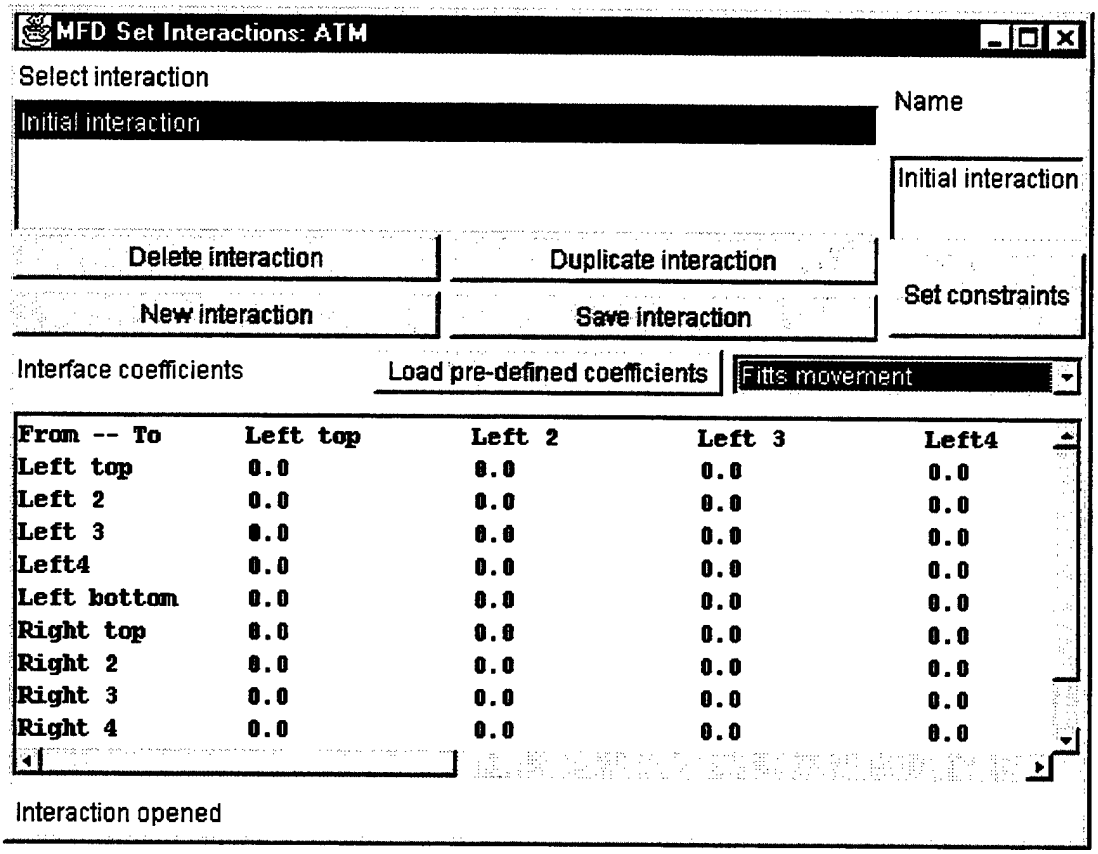


Figure 11: The `MFD Set Interactions` window as it appears after selecting the `Initial interaction`.

are the button names. The numbers then indicate the coefficient for a relationship from the button in a row to the button in the column. It is not necessary for interaction coefficients to be symmetric (that is the coefficient from *Left top* to *Left 2* need not be the same as the coefficient from *Left 2* to *Left top*).

Be default, a new interaction has coefficients of zero. The designer can edit the coefficients by simply changing the 0.0 numbers to whatever is desired. The description of a coefficient is simply text, so the designer may also import calculations of coefficients from other programs, as long as they are in the format required by MFDTool. The format is that each "cell" is separated by a tab, and there must be an item in every cell. This format is consistent with copying and pasting from many spreadsheet programs. The top row and left column cells are not interpreted by MFDTool (but there must be something in those cells). All other cells must be numbers. The status line at the bottom of the `MFDSet Interactions` window will identify the location and type of the first found error (if any) as it tries to parse the textual information.

MFDTool also provides mechanisms for computing a number of useful interaction coefficients. The choices are in the pull-down options just above the coefficient matrix box. To set the coefficients to be any of these, select the desired set of coefficients and then click on the `Load pre-defined coefficients` button. The coefficients matrix text box will immediately fill in with corresponding numbers. For example, Figure 12 shows the `MFD Set Interactions` window after loading coefficients that correspond to *Euclidean distance*. To save these interaction coefficients, click on the `Save interaction` button.

With the exception of the `Set constraints` button, the other parts of the `MFD Set Interactions` window are fairly straight forward. The name of the currently selected interaction can be changed in the `Name` text field. A new interaction can be created by clicking on the `New interaction` button. An interaction can be deleted with the `Delete interaction` button (but there must always be at least one interaction). An interaction and all of its properties (including constraints) can be duplicated with the `Duplicate interaction` button.

6.1 Setting constraints

After an interaction is created a designer can define constraints relative to that interaction. Select the desired interaction by clicking on its name in the list on the `MFD Set Interactions` window. Then click on the `Set constraints` button. A new window will appear that looks like Figure 13. The title of the window identifies both the name of the MFD and the name of the selected interaction.

Constraints are defined in the `MFD Constraints` window, with the `MFD Hierarchy`, `MFD Hardware`, and `MFD Page list` windows being optionally used to select various MFD pages for constraining.

The creation of a constraint begins by selecting the type of constraint that is desired. This is done with the menu choice option on the right side of the `MFD Constraints` window. After the type of constraint is selected, click on the `New constraint` button. What to do next depends on the type of constraint selected, as discussed below.

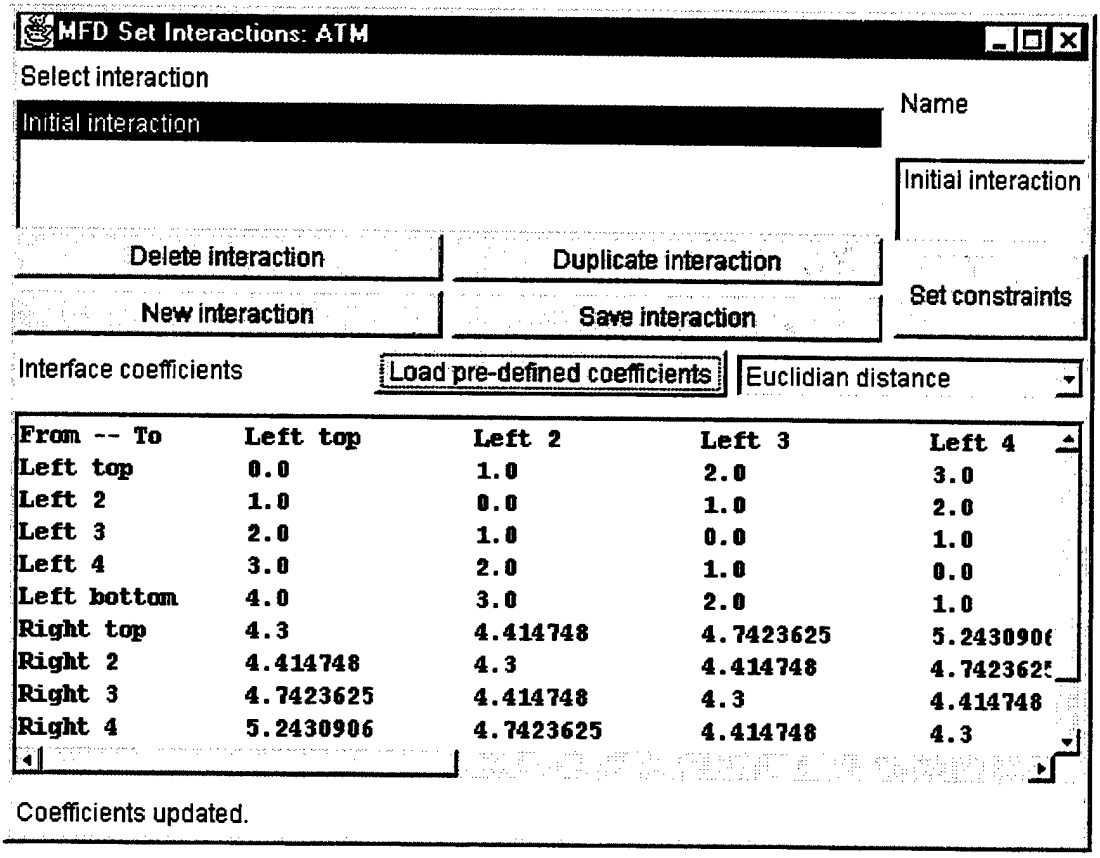


Figure 12: The `MFD Set Interactions` window as it appears after loading coefficients for *Euclidean distance*.

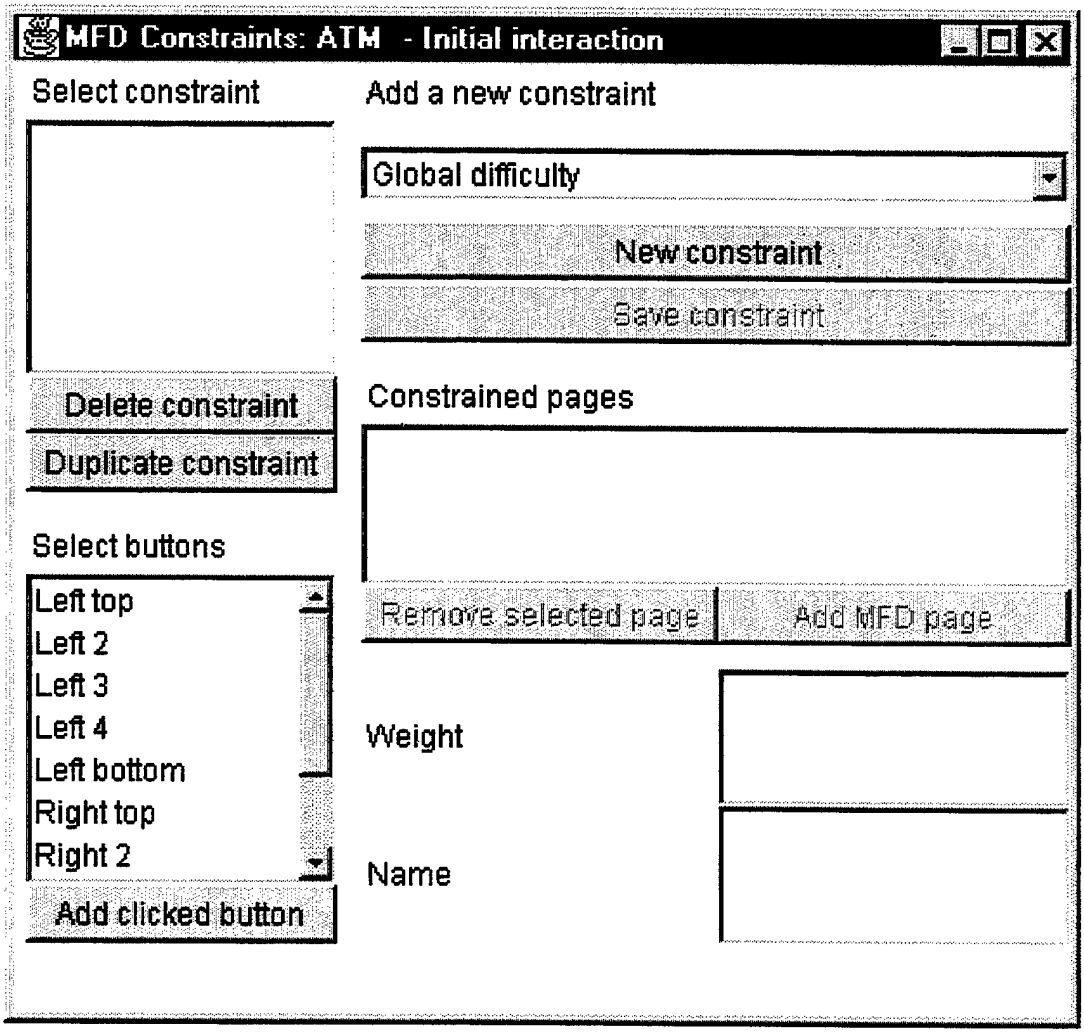


Figure 13: The `MFD Constraints: ATM - Initial interaction` window when it first appears.

6.1.1 Global difficulty

This constraint acts to minimize the average sum of interaction coefficients as the user goes through the button presses that traverse the hierarchy to access different MFD pages. The designer has the option of providing a name for this constraint (the default is *C#*, where # is the number of constraints currently defined). The name is listed in the text field on the bottom right of the `MFD Constraints` window. The designer also has the option of setting the weight for this constraint in the textfield directly above the name textfield; the default is 1.0. Once the name and weight are as desired, click on the `Save constraint` button. The list box on the left side of the `MFD Constraints` window now shows the name of the just saved constraint. Figure 14 shows what the `MFD Constraints` window looks like after the constraint is created and saved. There is no reason to include more than one `Global difficulty` constraint for an interaction.

In the list of constraints, the number to the right of the constraint name gives the cost of the current mapping of MFD labels to buttons. From the listing of constraints, selecting a constraint loads the selected constraint's properties into the appropriate constraint fields. These can then be edited and clicking on the `Save constraint` button updates the constraint. The `Delete constraint` button deletes the currently selected constraint. The `Duplicate constraint` button makes a copy of the currently selected constraint.

The optimization on this constraint considers the proportion of time each page is needed. The page proportions can be set at any time (e.g., before or after defining the constraint) on the `MFD Page list` window, as described above.

The set of constrained pages is the entire MFD hierarchy by default. Selecting the `Global` constraint will provide a listing of all MFD pages in the `Constrained pages` list box on the middle right of the `MFD Constraints` window. You can remove a page from this list or add a page to this list. Adding a page is done by selecting an MFD page from the `MFD Page list`, `MFD Hardware`, or `MFD Hierarchy` window and then clicking on the `Add MFD page` button in the `MFD Constraints` window. If the MFD page has already been assigned to this constraint, the status line at the bottom of the `MFD Constraints` window will indicate as much.

6.1.2 Pages to close buttons

This constraint has two main purposes. The first is to place specified labels on the same MFD screen as close together as possible. The second is to place specified labels on different MFD screens as close together as possible. "Closer" is defined as a smaller interaction coefficient. To create this type of constraint, select `Pages to close buttons` from the menu choice on the right side of the `MFD Constraints` window and then click on the `New constraint` button. Set the name and weight as desired. Assign selected pages, as described above. Figure 15 shows what the `MFD Constraints` window looks like after all the pages with the name `Cancel` are constrained to be close to each other.

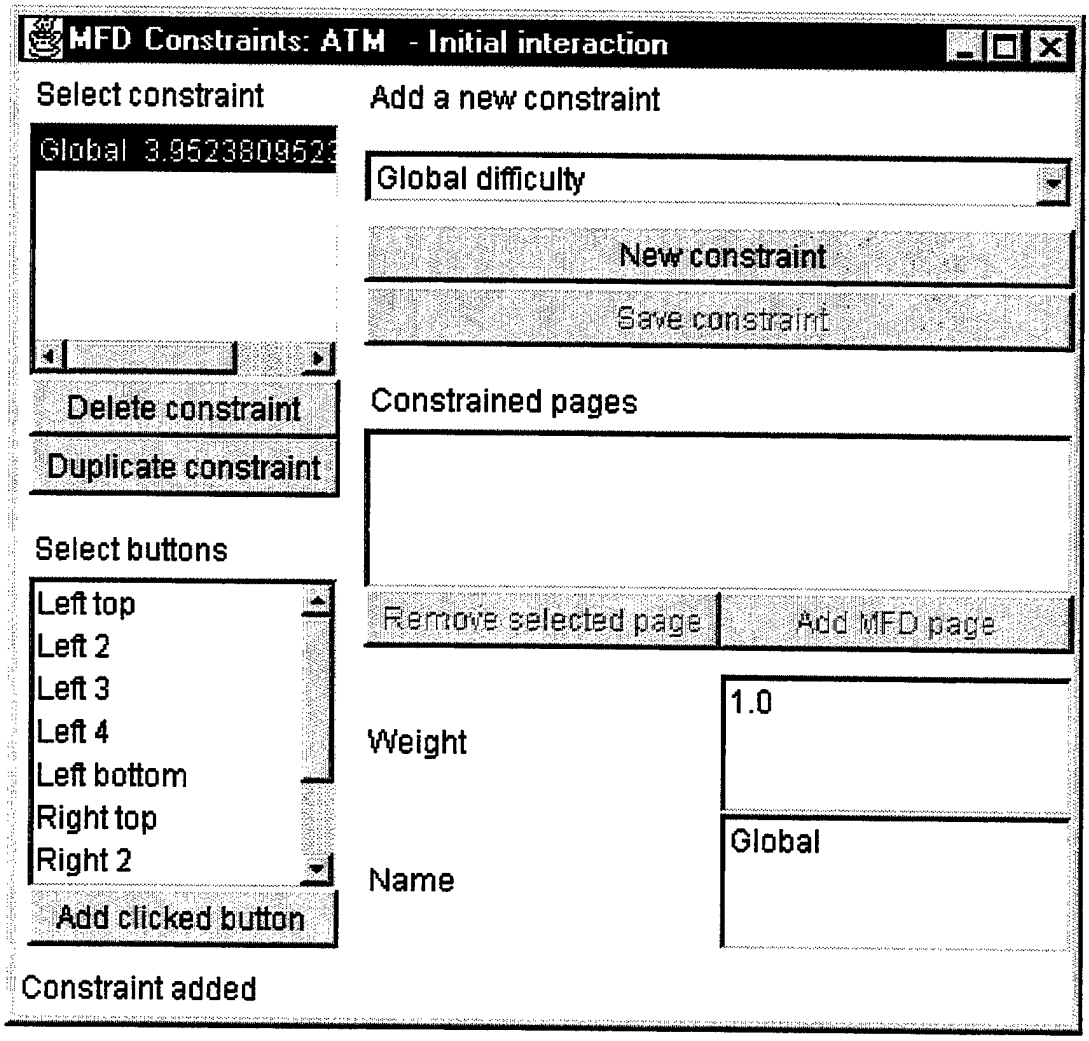


Figure 14: The `MFDTool` window as it appears after creating and saving the constraint `Global`.

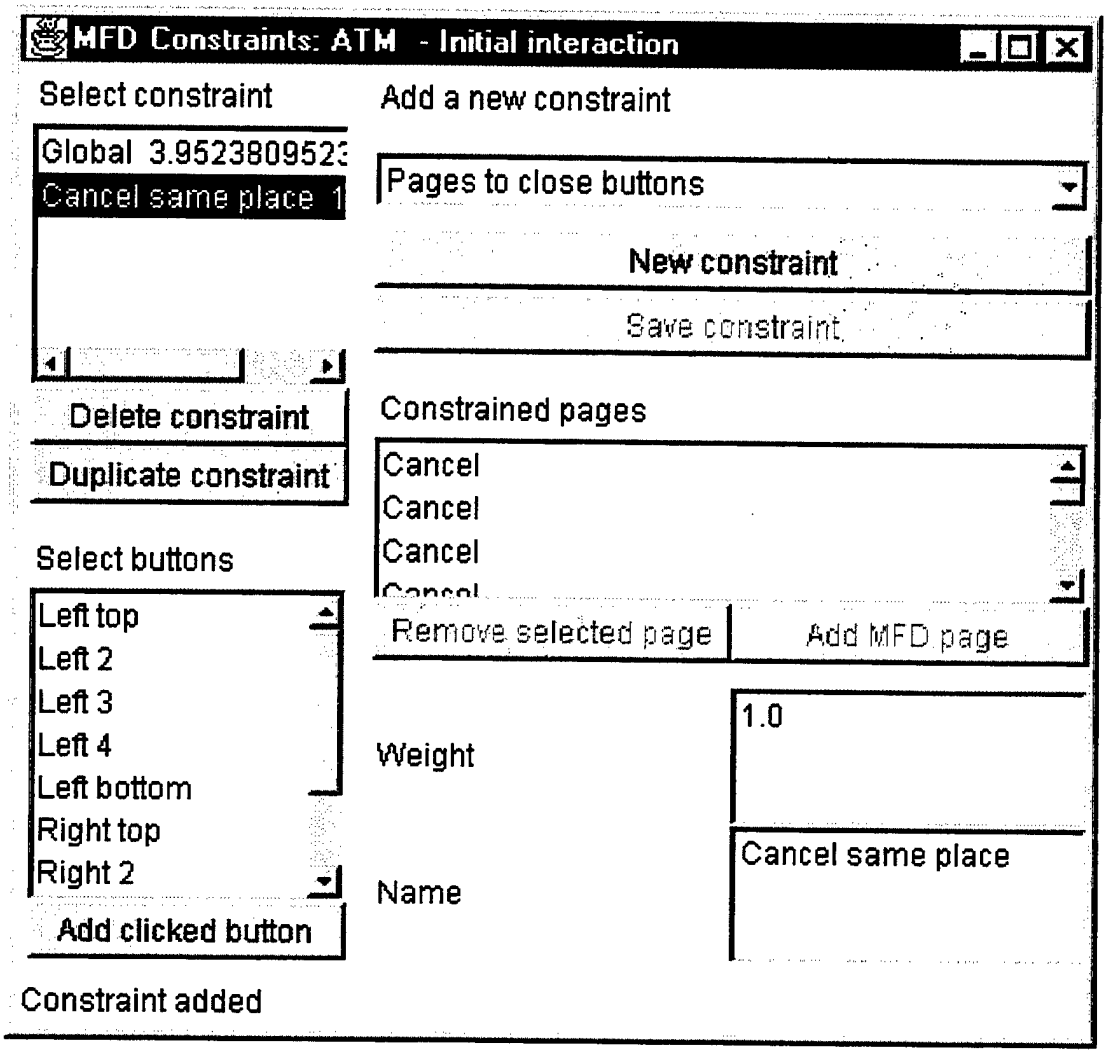


Figure 15: The `MFD Constraint` window as it appears after creating and saving the constraint `Cancel same place`. The list `Constrained pages` shows all the pages restricted by this constraint.

6.1.3 Pages to far buttons

This constraint is the complement of the previous one. It may be useful if a designer wants to insure that certain buttons are very separate; perhaps to avoid confusion by a user, or to avoid the risk of error.

Setting up the constraint is essentially the same as for the `Pages to close buttons` constraint.

6.1.4 Pages to fixed buttons

This constraint allows the designer to restrict specified page labels to specified buttons. To create this type of constraint, select `Pages to fixed buttons` from the menu choice on the right side of the `MFD Constraints` window and click on the `New constraint` button. Set the name and weight of the constraint as desired. Assign selected pages as for the `Pages to close buttons` constraint, above. In addition, from the list of `Select buttons`, click on the names associated with the buttons you want the pages to be restricted to. An alternative means of selecting a button is to click on the desired button in the `MFD Hardware` window and then click on the `Add clicked button` in the `MFD Constraints` window. This will automatically select the just clicked button.

Figure 16 shows the `MFD Constraints` window after the designer has introduced a constraint to place the `Cancel` labels to be on either of the two bottom buttons. Notice that in combination with the constraint to place all the `Cancel` labels on close buttons, the optimized MFD design should put all the `Cancel` labels on either the bottom right or the bottom left button.

6.1.5 Path difficulty

This constraint applies to a designer-defined special sequence.

To create a path difficulty constraint, select `Path difficulty` from the menu choice on the right side of the `MFD Constraints` window, then click on the `New constraint` button. Set the name and weight of the constraint as desired. The designer now needs to assign pages to the constraint *in the same order* that the user will travel through the pages along the "special path." Select the first page along the path, and click on the `Add MFD Page` button. For each subsequent page along the path, select that page and click on the `Add MFD Page` button. The order that the pages are added to the constraint is the order that MFDTool will assume the user will move through the hierarchy.

Once the path is defined, click on the `Save constraint` button. Figure 17 shows the `MFD Constraint` window after saving a short `Path difficulty` constraint.

6.1.6 Parent to child variability

This constraint differs in many respects from the other constraints. Those constraints minimize (or maximize) some function of the interaction coefficients. This constraint minimizes variability among the interaction coefficients for the buttons from a parent to a child. The designer specifies the child pages.

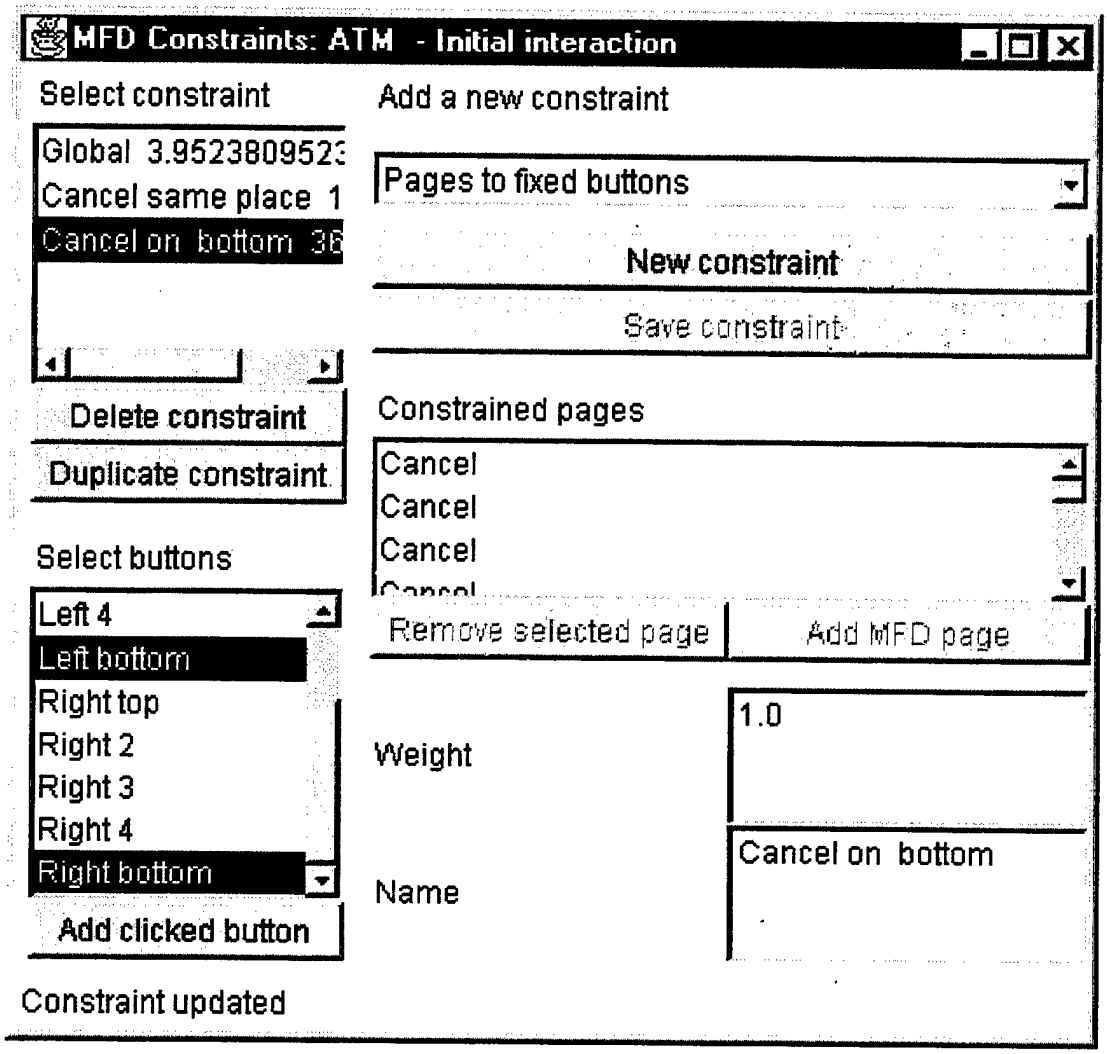


Figure 16: The `MFD Constraints` window as it appears after creating and saving the constraint `Cancel on bottom`. The list `Constrained pages` shows all the pages restricted by this constraint. The list `Select buttons` shows the buttons that the constrained pages must be restricted to.

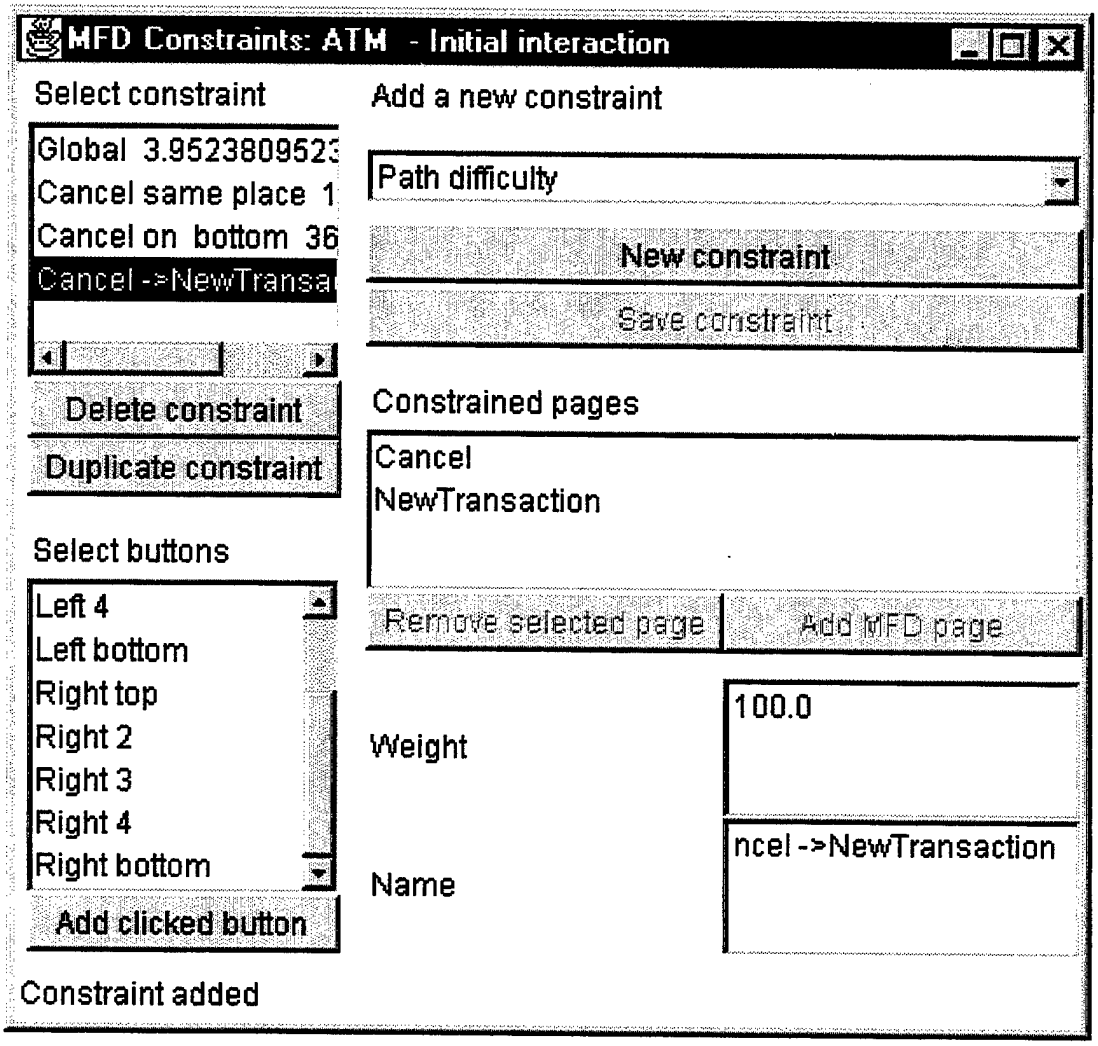


Figure 17: The `MFD Constraints` window as it appears after creating and saving the constraint `Cancel -> New Transaction`. The list `Constrained pages` shows the path of pages restricted by this constraint. In this case, the path starts at the `Cancel` hyperlink under `Fast cash` and goes to the `New Transaction` page under the `Cancel` page near the top of the hierarchy. MFDTool will assign page labels to buttons so that the sum of interaction coefficients is minimized along this path of pages.

For example, perhaps a designer wants each `Checking` label to be in the same position relative to its parent's label button. To create such a constraint, select `Parent to child variance` from the menu choice on the right side of the `MFD Constraints` window, then click on the `New constraint` button. Add the desired pages to the `Constrained pages` list (in this case, each page named "Checking"). Set the name and weight as desired, and then click on the `Save constraint` button. Figure 18 shows the `MFD Constraint` window after saving this constraint.

With the current *Euclidean distance* interaction, MFDTool will act to place each `Checking` label on a button at the same distance from it's parent's button.

7 Optimization

Once all the desired constraints are set, the designer can start the optimization process. Simply click on the `Start optimization` button on the `MFDTool` window. A new window will appear that looks like Figure 19.

Depending on your computer, the size of the hierarchy, and the complexity of the constraints, the optimization process may take a few minutes to several hours.

There is no way to adequately describe the effect of optimization in written text. A previously optimized design is included with MFDTool. Start MFDTool and go to `File`→`Open`. From the open file window, select the directory `Examples` and then the file `ATM1.mfd`. Clicking through the `MFD Hardware` will give a good idea of the assignment of page labels to buttons.

The optimization is, of course, relative to the defined constraints. For this file, the constraints were as described in the discussion of constraints above. Figure 20 shows the `MFD Constraints` window for the optimized design. Note that the costs for the different constraints are zero, except for the `Global` constraint. This means that all but the `Global` constraint are perfectly satisfied.

The resulting design deals with a number of subtle issues with regard to these constraints. For example, the `OK Password` page is on the middle left button. After selecting the `OK Passwords` page note that `Account info` is on the very same button. This is the best assignment, rather than placing some other page on that button because `Account info` is a parent page that contains many children underneath it. In terms of overall probability of access, the user will go through the `Account info` page more often than any other page. Thus, the interaction coefficients will be minimized by putting `Account info` on the same button as the `OK Password` page.

Notice that on each page, the labels tend to be clustered around the same button as was used to access the page. This minimizes the interaction coefficients.

Notice that the `Cancel` pages are all on the same button and on a bottom button.

Notice that the `NewTransaction` label is on the same button as the `Cancel` labels. This satisfies the `Cancel->NewTransaction` constraint.

Notice that a `Checking` label is always located on a button that is one above or one below the button that was used to get to the page containing that label. This minimizes the parent to child variance among these labels.

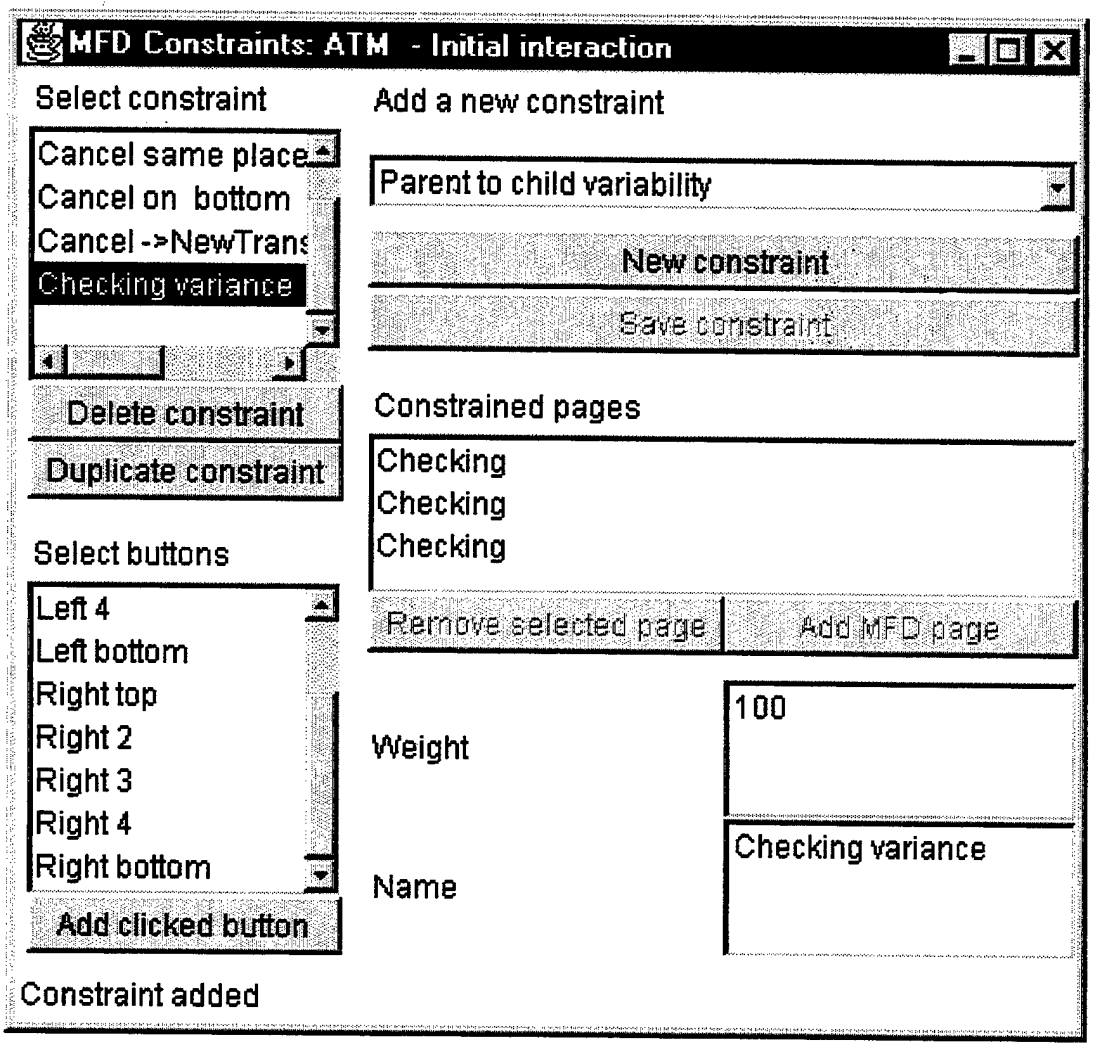


Figure 18: The `MFD Constraints` window as it appears after creating and saving the constraint `Checking variance`. The list of `Constrained pages` shows all the pages restricted by this constraint.

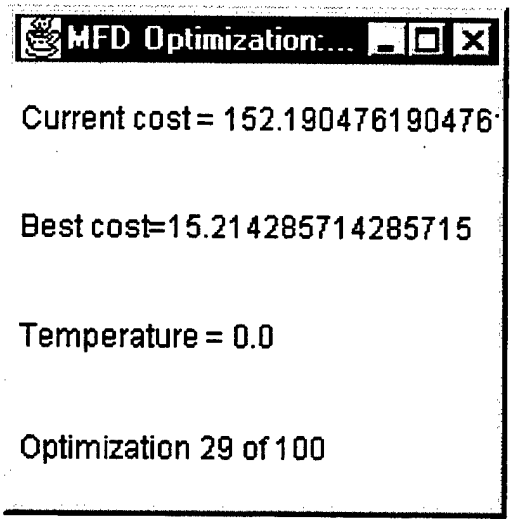


Figure 19: The `MFD Optimization` window as it appears while running an optimization.

In short, once the MFD hardware, hierarchy, and constraints are defined, the optimization is handled by the computer. The optimization is exceedingly thorough, and considers many details that a human designer will likely not have the time to deal with. Naturally, the design should still be validated by the human, as it is possible that the selected constraints do not correctly identify the important aspects of the design problem. It is also quite possible that the optimization process does not find the *absolute* best design, but only a pretty good design. Occasionally, a designer can look at the "optimized" design and recognize a better variation of what MFDTool provides. Alternative designs can be explored with the manual setting of button assignments.

7.1 Setting optimization parameters

MFDTool optimization starts with a default set of optimization parameters, but a designer can surely do better by setting things appropriately. These parameters can be set by clicking on the `Set optimizer` button from the `MFDTool` window. Figure 21 shows what this window looks like.

MFDTool can use either of two types of optimization techniques. The appropriate parameters to set are described below.

7.1.1 Gradient descent

- *Number of swaps per cycle*: A cycle is a set of random changes of page button assignments. This parameter establishes the number of changes in a given cycle.
- *Number of initialization cycles*: At the start of each optimization run, MFDTool randomizes the assignment of pages to buttons. This parameter sets how many cycles of random changes are made to define the random initialization state. Bigger numbers

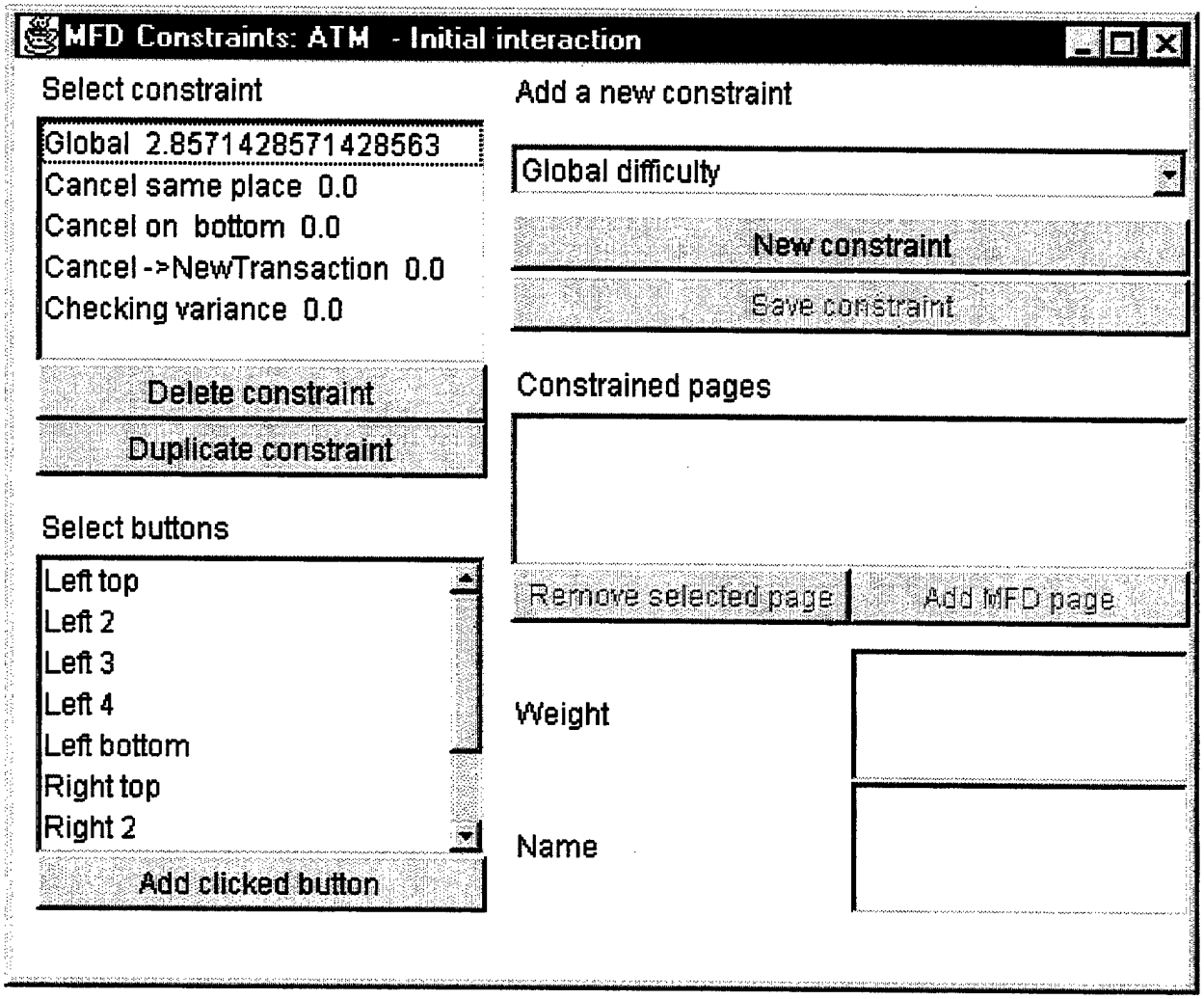


Figure 20: The `MFD Constraints` window as it appears after the optimization. The cost for all but the `Global` constraint is zero, indicating that these constraints are perfectly satisfied.

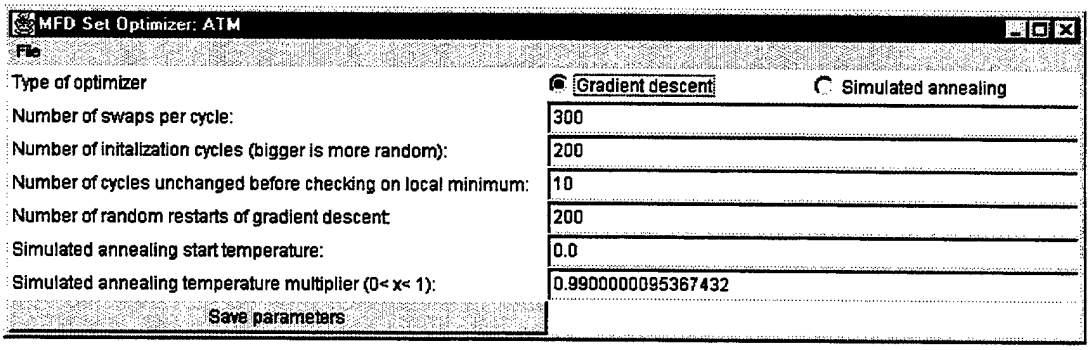


Figure 21: The `MFD Set Optimizer` window.

mean the random initial state is more random, which is generally better when searching for an alternative to a design that was just found.

- *Number of cycles unchanged before checking on local minimum*: When it seems that the optimization has finished, MFDTool checks to verify that there is no single change in button assignments that would produce a lower cost. MFDTool decides that it should check for a local minimum when a specified number of cycles has not produced any changes in cost. Setting this number very low likely means that MFDTool will often check before it should. Setting this number very high likely means that MFDTool will continue making random changes that cannot possibly improve the optimization. The terms "very low" and "very high" can only be defined relative to a specific optimization problem.

- *Number of random restarts of gradient descent*: Gradient descent is a fast optimization technique, but it often fails to find a global minimum of a cost function. To overcome this limitation, MFDTool can repeat the optimization from many different (randomly generated) initial states. This parameter sets how many times the optimization should be repeated. MFDTool keeps the best design from all of these optimizations.

- *Simulated annealing start temperature*: This parameter is not used by the gradient descent technique. Any value here will be ignored.

- *Simulated annealing temperature multiplier*: This parameter is not used by the gradient descent technique. Any value here will be ignored.

7.1.2 Simulated annealing

- *Number of swaps per cycle*: A cycle is a set of random changes of page button assignments. This parameter establishes the number of changes in a given cycle. The simulated annealing temperature is fixed during one cycle.

- *Number of initialization cycles*: At the start of each optimization run, MFDTool randomizes the assignment of pages to buttons. This parameter sets how many cycles of random changes are made to define the random initialization state. Bigger numbers mean the random initial state is more random, which is generally better when searching for an alternative to a design that was just found.

- *Number of cycles unchanged before checking on local minimum*: When it seems that the optimization has finished, MFDTool checks to verify that there is no single change in button assignments that would produce a lower cost. MFDTool decides that it should check for a local minimum when a specified number of cycles has not produced any changes in cost. Setting this number very low likely means that MFDTool will often check before it should. Setting this number very high likely means that MFDTool will continue making random changes that cannot possibly improve the optimization. The terms "very low" and "very high" can only be defined relative to a specific optimization problem.

- *Number of random restarts of gradient descent*: This parameter is not used by the simulated annealing technique. Any value here will be ignored.
- *Simulated annealing start temperature*: The start temperature needs to be large enough to allow lots of increases in cost to be accepted by the simulated annealing optimizer during the early stages of the process. Setting this number bigger generally makes the optimization better (to a limit), but it also increases the time required for the optimization to finish.
- *Simulated annealing temperature multiplier*: After each cycle, the simulated annealing temperature is multiplied by this number. Simulated annealing works by gradually reducing the temperature, and this parameter defines what "gradually" means. Setting this number closer to 1.0 generally makes the optimization better (to a limit), but it also increases the time required for the optimization to finish.

Because of its random nature, the simulated annealing may sometimes find a final design that is not as good as one of the designs it found during its random changes. MFDTool always keeps the absolutely best found design and reports that as the best design of the optimization.

7.2 Saving optimizers

It sometimes takes quite a bit of tweaking to find a set of optimization parameters that work efficiently with a given MFD design problem. A designer may wish to re-use the optimizer parameters from one MFD task for another MFD task. With this in mind, MFDTool allows the designer to save the parameters of a created optimizer and to load those parameters for another MFD design task. This can be done through the `File` pull-down menu on the `MFD Set Optimizer` window.

8 Examples

This section describes several ATM designs that are optimized relative to different types of interactions and constraints. All of the MFD files are in the folder `Examples` and are accessed through the `MFDTool` window.

Realize that none of these examples are meant to necessarily indicate good MFD designs. They are included here to demonstrate how the different constraints and interactions are interpreted by the optimization process. The resulting MFD is only a good design if the interactions and constraints accurately capture the important issues in the use of the MFD.

8.1 Fitts Law: `ATMFitts1.mfd`

This MFD design highlights the use of Fitts law for defining interaction coefficients. To that end, the corner buttons are set to have much larger Fitts sizes than their actual sizes. This

should make movements toward the corner buttons much faster than movement to any other buttons.

In addition, page proportions were set so that page $20 under `Fast cash` has a proportion of $p = 0.2$, thereby indicating that it is frequently used.

Predictably, the optimized design tends to place labels on the corner buttons. The only exceptions are for when a parent label is not on a corner button, in which case it is faster still to put the label on the same button as the parent.

8.2 Fitts Law: `ATMFitts2.mfd`

This design is a variation of the previous one, with the assumption that the Fitts sizes of the corner buttons are actually the same as their physical size. Notice the dramatic change in the overall design.

8.3 Fitts Law with consistency: `ATMFitts3.mfd`

This design is a variation of the previous one, but includes an additional interaction, *Euclidean distance*, that is used to keep sets of pages with a common name on the same buttons.

The optimized design satisfies all the `Euclidean distance` constraints, and then minimizes the Fitt's law constraints as best as possible.

8.4 Fitts Law with relative consistency: `ATMFitts4.mfd`

This design is a variation of the previous one, but instead of requiring that all the `Cancel` pages be on the bottom and at the same buttons, it requires that they have a minimization of parent to child variance. This means that on each page, the `Cancel` label should be in a particular position relative to the button that was pressed to reach that page.

This restriction is imposed with the definition of two additional interactions: `X-directed` and `Y-directed`. For each of these interactions, all the `Cancel` labels are constrained to be placed on buttons that have the same interaction coefficient from their parent page button.

The optimized design is substantially different from the earlier designs. From a visual standpoint it may seem poor design to place the `Cancel` label on different buttons for different pages. However, this constraint imposes a commonality of movement to reach the `Cancel` button. Upon access of any MFD page, the user would move up one button to reach the `Cancel` button. In some situations, this consistency of movement may be a more important consistency than physical location.

8.5 Fitts Law with relative consistency and order: `ATMFitts5.mfd`

This design is a variation of the previous one, but it now addresses a potential conflict between the consistency and visual grouping of labels. In the previous design, the list of labels on the `Fast cash` page are scattered. This allows the `Cancel` button to be located one button above the `Fast cash` button. However, the cost in terms of visual arrangement of the monetary labels may be detrimental.

To address this issue, two new constraints are added to the `Euclidean distance` interaction. These constraints require the labels `$10` and `$20` to be next to each other and the labels `$20` and `$50` to be next to each other. The upshot of these constraints is that the labels `$10`, `$20`, and `$50` need to be in an order (either ascending or descending) for these constraints to be minimized.

Satisfying these constraints leads to an optimized design that is substantially different. It tends to meet these constraints at a cost to the `Global` constraint in the Fitts interaction (which has a smaller weight). In `ATMFitts5.mfd` the cost for the `Global` constraint is 447.06, while for the design in `ATMFitts2.mfd` the cost is 339.88. The difference for the former is that additional constraints are imposed, and the optimization trades off one cost for another.

8.6 Tabbing: `ATMTabbing1.mfd`

This design is a variation of the one in `ATMFitts3.mfd`. It replaces the Fitts movement interaction with one that specifies movement based on tabbing between buttons with arrow keys. In the `Tabbing` interaction it is assumed that an up arrow press at a top button loops down to the bottom button on the column. Likewise, a down arrow press at a bottom button loops up to the top button on the column.

8.7 Fitts Law and Tabbing: `ATMTabbing2.mfd`

This design includes a `Global` constraint for both a Fitts Law interaction and a Tabbing interaction. This could be useful if there are two types of possible interactions and the MFD needs to be designed for use by either one. Or, this could be a case where two different users must access the same MFD system, but will have different interfaces (e.g., pilot and copilot). Because the interaction coefficients for Fitts Law are numerically larger than those for the Tabbing interaction, the weights on the Tabbing constraints were increased.

8.8 Fitts Law and button occlusion: `ATMOcclusion1.mfd`

This design is a variation of the one in `ATMFitts3.mfd`. It modifies the Fitts movement interaction to consider the possibility that when a user has their finger over a button that many button labels are occluded. If the user must read the button labels to know where to move next (e.g., the user is not highly practiced and does not have the button assignments memorized), then the user may need to move their hand out of the way before making a movement to a desired button. This extra movement will take time, which is assumed to be a constant 1500 milliseconds for any time the occlusion makes a difference. (Note, these interaction coefficients are not making any assumption about how long it takes to actually read the labels, they simply assume that the *movement* time involved in moving the hand away from the screen and then moving it to the correct button is a constant 1500 milliseconds.)

The resulting design is quite interesting. MFDTool assigns labels to buttons to avoid possible occlusion. Although visually unappealing, in terms of use, this could be a very practical design.

8.9 Fitaly: `Fitaly.mfd`

This example shows an entirely different type of MFD, part of a keypad entry system for a Palm Pilot hand-held computer. Because a Palm Pilot is usually used with one finger (or a pen) instead of the ten finger technique used on typewriter keyboards, there is reason to expect that a new layout of keys can make for faster entry of information. The Fitaly keyboard was designed by Textware Solutions (`http://www.twsolutions.com`). The engineers at Textware Solutions based their design on consideration of the frequency of using individual letters and the frequency of letter-to-letter transitions. The former frequencies are published on their web site, and were used to create an analogous design in MFDTool. Interestingly, MFDTool creates a design quite similar to the Fitaly design. In particular, the six letters in the middle three rows of the middle two columns are the same in both designs. These correspond to the most frequently used letters.

The discrepancies between the two designs probably correspond to the effect of letter-to-letter transitions, which the current MFDTool example does not consider (but could be implemented with multiple `Path difficulty` constraints). Without consideration of those transitions, the MFDTool design is slightly more optimized relative to a Fitt's Law interaction. The MFDTool design has a global constraint cost of 182.77, while the Fitaly design has a global constraint cost of 185.85 (see the file `origFitaly.mfd`).

8.10 Alternatives to Fitaly: `Fitaly2.mfd`, `Fitaly3.mfd`, and `Fitaly4.mfd`

MFDTool can also be used to consider optimization of other hardware designs. For example, in `Fitaly2.mfd` the two space buttons are merged into a single large button and placed slightly off the center of the screen, the optimized assignment has a larger average Fitt's movement time (184.28) than the optimized layout for the hardware used by the original Fitaly system.

If the space buttons are merged and placed on the lower left as in `Fitaly3.mfd`, the optimized design produces a global constraint cost of 181.35. This is a difference of 1.5 milliseconds faster than the optimal design with the original Fitaly hardware configuration. It is 4.5 milliseconds faster than the Fitaly layout. A 4.5 millisecond difference is small, but it accumulates over extended use. MFDTool also identifies where the "home" for the user should be by assignment of the `Start` page to a button (row three, column 3; the *e* button).

Even better results are found for the design in `Fitaly4.mfd`, where the space button is one row high and four columns long and placed on the middle bottom of the display. The Fitt's Law interaction's global constraint for the optimized design is 167.61 milliseconds, which is 13.74 milliseconds faster than any other design discussed so far. Relative to the original Fitaly layout, this design has an 18.24 millisecond advantage.

Of course, these alternatives to the original Fitaly design may not be as good as the original if the transition probabilities between letters are also considered. Also one would want to experimentally validate the predicted differences in the designs.

9 Conclusions

MFDTool provides a means of optimizing the association of MFD hierarchical information with MFD buttons. Creating an effective association is difficult because of the large number of variables involved in the task. MFDTool quantifies the variables, thereby allowing standard optimization approaches to be applied to the problem.

MFDTool should rarely be used in isolation from the rest of the design process. Instead, it will probably be most beneficial as a collaborator with a designer. That is, a designer may create one hierarchical organization and hardware configuration and then run MFDTool to find the optimal layout of information. From that starting point, the designer can consider the effect of changing the hierarchical organization or hardware configuration, and re-running MFDTool for each change. The only fair comparison across different hierarchical or hardware configurations is relative to their optimal association. Such comparisons would have been nearly impossible in the past because the association itself was very difficult to optimize. MFDTool greatly simplifies this process, thereby allowing the designer to compare a larger set of possible designs.