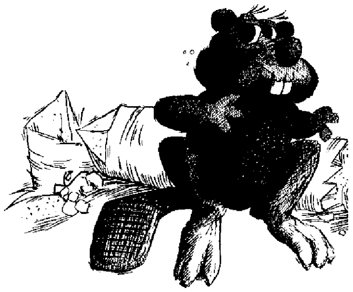


# *Learning Considerations in User Interface Design: The Room Model*

*Patrick P. Chan*
*Software Portability Laboratory*

*CS-84-16*

*July 1984*

# LEARNING CONSIDERATIONS IN USER INTERFACE DESIGN: THE ROOM MODEL[†]

*Patrick P. Chan*

Software Portability Laboratory
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

## *ABSTRACT*

*Room* is an environment in which users organize and issue computer system commands. Metaphorically, *Room* is based on the concept of a *room*, in contrast to systems incorporating the currently popular *desktop* model. Both the room and desktop models embody design principles that address the issues of reducing the overhead of learning an application. However, *Room* demonstrates techniques that improve an expert's productivity without compromising the principles embodied in the desktop model. We describe *Room*'s user interface, design, and implementation as well as the fundamental differences between the room and desktop models.

An approach is proposed for developing a framework wherein user interface design principles can be meaningfully analyzed, classified, and, most importantly, derived. We begin the specification of the framework with precepts from learning psychology and from it derive several design principles, many of which are popular in the literature. Each principle is presented with strategies that prescribe ways in which the principle can be implemented.

[†]This report was originally submitted as a Master's thesis to the Department of Computer Science, Faculty of Mathematics, at the University of Waterloo by the author.

# Acknowledgements

I would like to thank my supervisor, Prof. Michael A. Malcolm, whose encouragement and support were crucial to the development of this work. His fine editing skills greatly improved the readability of this report.

I would like to thank my thesis readers, Prof. Doug Dyment and Prof. Randy G. Goebel, for their valuable suggestions.

Several people have contributed to this work with their helpful comments. They are: Paul Bernard, Bert Bonkowski, Claes Harvenberg, Rosanna Lee, Ann Lo, Mark Maxted, Quentin Miller, Gary Stafford, Ian Telford, and R. Vasudevan.

I am also grateful for the financial support of the Department of Computer Science at the University of Waterloo.

# Table of Contents

# List of Figures

# Table of Principles

# 1

# Introduction

*"If we intend a science of human-computer interaction, it is essential that we have principles from which to derive the manner of the interaction between person and computer. It is easy to devise experiments to test this idea or that, to compare and contrast alternatives, or to evaluate the quality of the latest technological offering. But we must aspire to more than responsiveness to the current need. The technology upon which the human-computer interface is built changes rapidly relative to the time with which psychological experimentation yields answers. If we do not take care, today's answers apply only to yesterday's concerns."*

– Donald Norman†

## 1.1 Motivation and Objectives

The literature lists many principles for designing user interfaces. The approach for producing such principles has generally been based on retrospection and reflection during development; ideas that work in practice are generalized as much as possible and enunciated as principles. However, as Norman's observations convey, principles driven by current concerns suffer the consequences of being dated. Moreover, the excessive number of factors that influence a design makes it uncertain whether a *set* of principles evolved from one circumstance can be readily applied to another. The lack of a universal framework with which principles can be analyzed makes it impossible to relate principles not only between sets, but also within a set.

Our approach for producing a set of design principles is to examine the more stable side of the human-computer interface – human behavior. We have taken prevalent theories from the field of psychology and interpreted them in the context of user interface design. The result is a set of principles that is calculated to improve a user's productivity; phenomena that enhance performance are encouraged, while those that degrade performance are discouraged. The fundamental tenet of this approach is that as technology changes, the comparatively stable foundation of psychological precepts can always be reinterpreted to yield applicable

† "Design Principles for Human-Computer Interfaces", *CHI'83 Proceedings*, December 1983, pp. 1.

ideas. This approach also suggests a methodology for constructing a much needed framework in which to analyze, classify and derive user interface principles.

Of course, principles cannot approach their beneficial potential without examples of their application. We have incorporated these principles in the design of *Room*, an environment wherein a user organizes and issues his commands to the computer system. The user interface of *Room* and other popular command interpreters are analyzed to illustrate our set of principles.

Unfortunately, the scope of this report precludes a comprehensive development of a framework. We only demonstrate the methodology and potential for constructing such a framework. Also, we do not cover psychology at a depth it deserves. We focus only on the learning aspects of human behavior and not, for example, on personality or motivational aspects. Even within this restricted domain, we can only account for those psychological precepts that are immediately applicable to the design of productive interfaces.

At the very least, our interpretation of the theories of learning can be taken as plausible explanations for the phenomena we observe as user interface designers. By relating our observations to those of psychology, we can minimize the amount of redundant work and perhaps contribute to the development of psychology. We hope that the collaboration of these two disciplines – psychology and computer science – will form a synergistic relationship that inspires in each the grace of accelerated growth.

## 1.2 Organization

Chapter 2 provides a brief overview of a few popular command interpreters that are referred to in the ensuing chapters. The overview concentrates on the prominent characteristics of their user interfaces and on underlying principles which were applied in their design.

Chapter 3 presents the user interface of the *Room* environment in detail. This presentation concentrates on how to use *Room*, not on any design decisions or implementation details.

What methods can the designer employ to accelerate a novice's understanding of his system? Chapter 4 examines some theoretical models of mental processes which occur during learning. Our goal is not only to find ways to ease the mental effort that accompanies learning, but also to capitalize on this knowledge, once learned.

Chapter 5 presents the implementation design of the *Room* environment on Port. We first briefly describe the Port process model and then discuss *Room*'s process structure and the duties of each process.

In Chapter 6, we summarize the contributions of this report by (1) pointing out the fundamental differences between the desktop and room models, by (2) suggesting the immediate directions for pursuing and using the design principles framework and by (3) prescribing various enhancements that can be made to *Room*.

# 2

# Background

This chapter provides a brief overview of three popular operating systems – Unix†, Xerox Star and Waterloo Port††. We concentrate on the prominent characteristics of their user interfaces, especially those of their command interpreters, as this topic is used by the discussions in Chapter 4.

Unix was chosen because it represents the most sophisticated system in its class – the class of command line interpreters. Star offers one of the most innovative designs; its principles demonstrate the power of software integration and user-friendliness. Many contemporary operating systems have incorporated its design principles into their user interface. Port represents an interesting hybrid of the capability of Unix and the friendly qualities of Star.

As the discussions are very brief, the interested reader is directed to the reference list at the end of this chapter for detailed information about the systems.

## 2.1 Unix

Unix was developed starting in 1969 by Bell laboratories and has since become one of the most popular operating systems available. The early versions of Unix ran on DEC's PDP mini-computers series with teletypes as the main display devices.

Unix supports a very powerful command interpreter called the *shell.* The fact that teletype-like devices were the main display devices (and still are) strongly influenced the design of the shell throughout its evolution. Even in its latest, most advanced form, the *csh* [Joy 1979], all that is required to use it is a simple teletype. Consequently, almost any ASCII terminal will function with Unix. We will briefly describe the main features of the shell.

### 2.1.1 Pipes

Output from a command can be input to another command with the use of a *pipe.* A *pipeline* consists of two or more commands connected by pipes. A command within a pipeline acts like a filter whose output is a transformation of its input. Many complex commands can be formed with a pipeline of simple commands.

†Unix is a trademark of Bell Laboratories.
††Waterloo Port and Port are trademarks of the University of Waterloo.

### 2.1.2 Concurrency

When a command is executed, the shell is no longer available until the command is completed. Commands which may take a long time can be made to execute in the *background*, freeing the shell to accept other commands. Several commands can execute in the background simultaneously. Unfortunately, if several commands produce output, the results on the display may be unintelligible. The user may stop or destroy any background command at any time. A command executing in the background can be brought to the *foreground* if, for example, it requires input from the user.

### 2.1.3 The Shell as a Language

One of the most powerful features of the shell is that it can function as an interpreter. The shell supports iterative and conditional control flow constructs which can be used to build quite elaborate commands. The shell also supports variables, expression evaluation and even recursion. Programming with ordinary programming languages can often be avoided by writing a shell program. This feature allows non-programmers and users without access to system code to tailor system commands to a certain degree.

### 2.1.4 The History Command

The shell's *history* command produces a numbered list of the most recently executed commands available for easy re-execution. The desired command can then be retrieved by either number or pattern. This is one of the shell's features which attempts to reduce the amount of typing necessary to execute commands.

The shell also permits a previous command to be changed before it is executed. First, a command is selected as if it were to be re-executed; then, on the same line, changes to the command are specified using an editor-like syntax.

### 2.1.5 Text Substitution

There are three more ways you can reduce your typing load in the shell.

• **Alias Substitution** enables a command to be renamed, or parts of a command line to be reduced to a single word. This feature is useful to shorten frequently used command lines or to rename command names to something more easily remembered. Aliasing also applies to commands within pipelines.

• **Variable Substitution** enables you to use the value of a predefined variable in constructing commands. Variables can be changed at any time and retain their values for the duration of the session.

• **Filename Substitution** enables you to specify a file or a set of files with a pattern. If the pattern does not uniquely identify a file but rather a set of files, then the pattern will be substituted by the entire set. Hence the command will operate on each file in the set.

## 2.2 Xerox Star

Xerox introduced its 8010 Star Information System personal computer in 1981. It was designed for use in an office environment, supporting capabilities such as document preparation (complete with graphics), data processing, electronic filing, mailing and printing. Each workstation has networking capability, facilitating such services as remote file storage and mail. Two notable hardware features are its high resolution bit-mapped display and its mouse. The display can render high quality graphical objects, enabling documents with graphical and mathematical content to be viewed and edited without the need to first phototypeset it. The use of the graphics capability extensively pervades Star's user interface. The mouse is an essential component of the system in that the user interface relies on it heavily to reduce the amount of typing required of the user.

### 2.2.1 Principles

Eight principles guided the design of Star's user interface.

- Familiar user's conceptual model
- Seeing and pointing versus remembering and typing
- What you see is what you get
- Universal commands
- Consistency
- Simplicity
- Modeless interaction
- User tailorability

The most prominent of the principles are the first three. With the first, designers have creatively transformed many common computer operations into natural procedures that novice users easily understand. They have achieved this by taking advantage of knowledge a person possesses *before* encountering the system. The objects in the system closely resemble, in both form and behavior, familiar physical objects in the office. For example, there are folders, file cabinets, mail boxes, printers and note pads; the graphics help generate convincing reproductions. These objects are also manipulated in ways similar to their physical counterparts. For example, to mail a document, you *pick up* the document and *move it* into an outgoing mailbox.

The model of familiar objects is unified by a desktop metaphor. The objects appear as small *icons* on the desk and can be *opened* from the desk. An icon opens into a larger form called a *window* which enables its contents to be displayed and modified.

With the second and third principles, Star attempts to reduce many common commands into sequences of mouse movements. Many tasks are possible simply by moving icons about with the mouse. In this way, you feel that you are accomplishing the task yourself instead of trying to give a command to the computer. This eliminates one level of control and many problems which arise because of it. There are no special keywords to remember and type, only the application of natural procedures using the mouse. Also, Star reduces the amount of information a user must remember by making objects and commands visible. There is always a list of possible commands in a window. Not only does this make the system easier to learn, it makes it easier to start commands. You need only point to the place where the command is displayed and press a button on the mouse. Again, there are no keywords to remember and type.

### 2.2.2 Property Sheets

Most objects in Star have *properties* which determine how they look or behave. For example, the properties of a character include its font, size, and face (bold, italics, underline). The properties of an object are examined by selecting the object and pressing the SHOW PROPERTIES key. A *property sheet* window appears which displays all the available properties the object can assume and the properties currently in effect. The property sheet also permits the properties to be changed.

Property sheets prevent a user from being overwhelmed with the numerous possibilities of the system. Information is provided only when requested. But more importantly, the source for information is always known once the properties concept is understood.

## 2.3 Port

The development of Port began in 1980 as an ongoing research project to examine various aspects of network operating systems and user interface design. Port's networking capability permits resource sharing and communication between workstations. Port currently runs on various machines, the most popular being the IBM personal computer with a character-oriented display.

The Port user interface is designed to be used with or without a mouse. A mouse can greatly reduce the amount of typing, making Port especially productive for non-touch typists. However, there are several convenient mechanisms in the user interface which make most interactions on a mouseless workstation just as fast as on a mouse-endowed workstation.

### 2.3.1 Windows

The Port user interface supports multiple windows and concurrency, enabling several programs to be executing simultaneously, and viewed simultaneously. Each window occupies the full width of the screen. Windows can grow and shrink vertically but cannot overlap. There can be many windows; the screen displays as many as will fit. Those that are *hidden* can easily be brought into view. Figure 2-1 shows a Port display with four windows. *Edit* and *Browse* are visible while *Room* and *Message* are hidden. *Room* or *Message* can be made visible by selecting its respective activity name.

Collectively, windows behave like a stack. Whenever a hidden or new window is brought into view, the visible windows are forced "down" far enough to accommodate the hidden or new window. Windows at the "bottom of the stack" which no longer fit on the display are hidden. A partially hidden window is not allowed; a window disappears if it cannot be entirely displayed. The maximum window size is constrained by the size of the display.

An application makes its services available through a window by displaying a set of operations near the window's banner (see Figure 2-1). The desired operation can be selected with either a mouse or a function key on the keyboard. Operations typically operate on objects contained in the window. This type of interaction

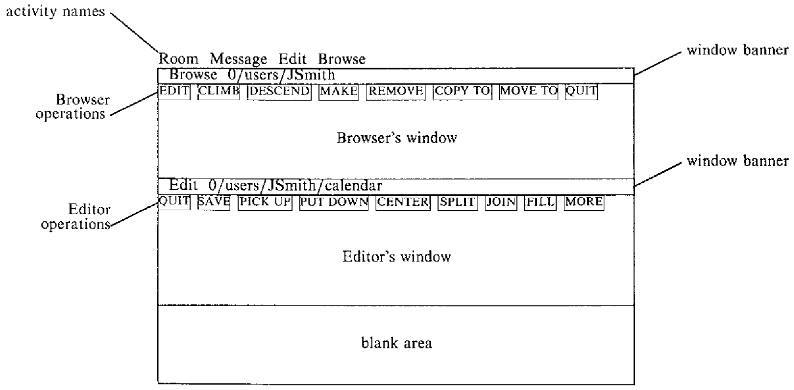


**Figure 2-1:** Port Windows

greatly reduces the need for typing as both objects and operations can be specified by merely pointing to them and pressing a button on the mouse or keyboard.

### 2.3.2 The Command Interpreter

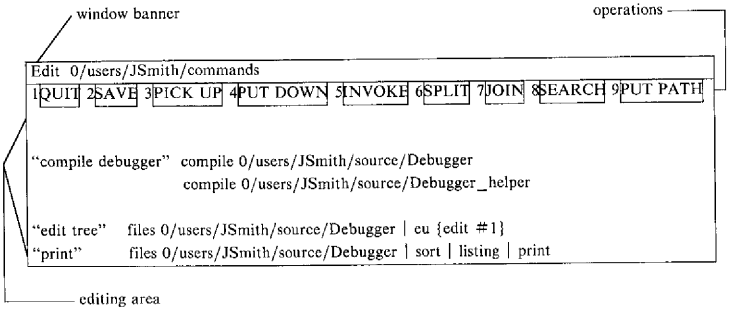
Port allows Unix-like commands, as well as pipes and redirection of standard input and standard output.

Commands are typed and executed from within the Editor. There are several advantages with this design. The power of the screen editor to edit text is available for command editing. By keeping files of commands in strategic locations in the file system, commands seldom need to be typed. This obviates Unix's history, aliasing and command correction features. A command is executed by selecting the line containing the desired command and selecting the INVOKE operation in the Editor (see Figure 2-2). A window then appears which executes the command. A group of consecutive commands can be executed in sequence by selecting the set of lines containing the commands and then selecting INVOKE.

Commands may also be executed from *Room*. *Room* and aspects of Port's user interface are discussed in greater detail in the following chapter.

### 2.3.3 The Browser

The *Browser* is an application program that provides file system services which allow files to be conveniently examined, copied, moved and removed. It also enables you to easily "move around" either a local or remote file system tree, maintaining an up-to-date display of the files below the current file. The display is immediately updated if a file is created or removed under the current file. Several



```
Edit 0/users/JSmith/commands

"compile debugger"  compile 0/users/JSmith/source/Debugger
                    compile 0/users/JSmith/source/Debugger_helper

"edit tree"    files 0/users/JSmith/source/Debugger | cu {edit #1}
"print"        files 0/users/JSmith/source/Debugger | sort | listing | print
```

**Figure 2-2:** The Port Commands Editor

Browsers may be active, enabling a user to view parts of one or more file systems simultaneously.

Only those Browser operations that use a new filename require typing. In addition, the Browser makes the name of a selected file available to other applications.

## 2.4 Summary

We have discussed the user interfaces of three popular operating systems. All three have been designed for different hardware configurations. Unix runs with simple teletype-like devices, Port with character-oriented displays and Star with high-resolution bit-mapped displays.

The Unix shell is functionally superior to the Port and Star command interpreters mainly because of its interpretive programming capabilities. It provides many convenient features which make interaction with a simple teletype both bearable and productive.

The major contributions of Star are design principles exploiting concepts of familiarity and visibility which decrease the amount of training a user needs in order to become proficient with a system.

Port captures most of the functionality of Unix and tempers it with qualities of Star. With extra capability in the display device, it is able to transform many of the convenient Unix features into an even more productive environment.

## 2.5 References

[Dolotta and Mashey 1979]
T.A. Dolotta and J.R. Mashey, "Using a Command Language as the Primary Programming Tool", *Command Language Directions*, Proceedings of the IFIP TC2.7 Working Conference on Command Languages, Berchtesgaden, West Germany (Sept. 1979), pp. 35-49.

[Joy 1979]
W. Joy, "An Introduction to the C shell", *Unix Time-Sharing System: Unix Programmer's Manual*, Seventh Edition, Volume 2B (Jan. 1979).

[Malcolm et al 1983]
M.A. Malcolm, P.A. Didur and P.A. McWeeny, *Waterloo Port User's Guide*, Software Portability Group, University of Waterloo, Dec. 1 1983.

[Malcolm and Dyment 1983]
M.A. Malcolm and D. Dyment, "Experience Designing the Waterloo Port User Interface", *Proc. of Small Computers*, San Diego, CA (Dec. 1983), pp. 168-175.

[Ritchie and Thompson 1974]
D.M. Ritchie and K. Thompson, "The Unix Time-sharing System", *Comm. ACM*, Vol. 17, No. 7 (July 1974), pp. 365-375.

[Seybold 1981]
J. Seybold, "Xerox's Star", *The Seybold Report*, Seybold Publications, Vol. 10, No. 16 (April 1981).

[Seybold 1981]
J. Seybold, "The Xerox Star", *The Seybold Report on Word Processing*, Seybold Publications, Vol. 4, No. 5 (May 1981).

[Smith et al 1982]
D.C. Smith, C. Irby, R. Kimball, B. Verplank and E. Harslem, "Designing the Star user interface", *Byte*, Vol. 7, No. 4 (Apr. 1982), pp. 242-282.

# 3

# The *Room* Interface: Rooms and Icons

*Room* is a special window in which a user organizes and issues commands to the computer system. The user's environment is conceptually modeled as a single-story building that houses many *rooms*. In each room there are *icons* that are used for various tasks. Travel to another room is done by passing through a *door*, represented by an icon. Figure 3-1 illustrates the visual representations of these objects. The bottom of the display shows the name of the room and the name of the user using the computer. The depicted room is JSmith's Studio.

Four icons occupy JSmith's Studio – two *door icons* and two *activity icons*. The name of a door icon indicates the room to which it leads. One door icon leads to JSmith's Gallery and the other to his Office. An activity icon is used to start an application. The name and picture on an activity icon describe the application it represents. The activity icon named "Paint" starts a paint application while the icon named "Mail" sends and retrieves mail.

## 3.1 Manipulating Icons

*Room* provides several operations that can be used on icons. Before an icon can be used in an operation, it must first be *selected* to distinguish it from other icons. An icon can be selected by pointing to it with the cursor and then pressing the SELECT key; the selected icon is highlighted. Pointing can be done with either the CURSOR KEYS or the mouse.

The TAB key can also be used to select an icon. Successively pressing TAB causes each icon to be selected in succession from left to right, top to bottom. Shifting the TAB key, reverses the order of selection. This selection method is preferable to manipulating the cursor if the workstation is not equipped with a mouse.

Icons can be rearranged in a room. Repositioning the selected icon is achieved by pointing to the desired vacant location and pressing the POSITION key. The selected icon moves to the new position. Moving the selected icon to another room is achieved by pointing to a door and pressing POSITION. This causes the selected icon to move through the door, into the adjoining room.

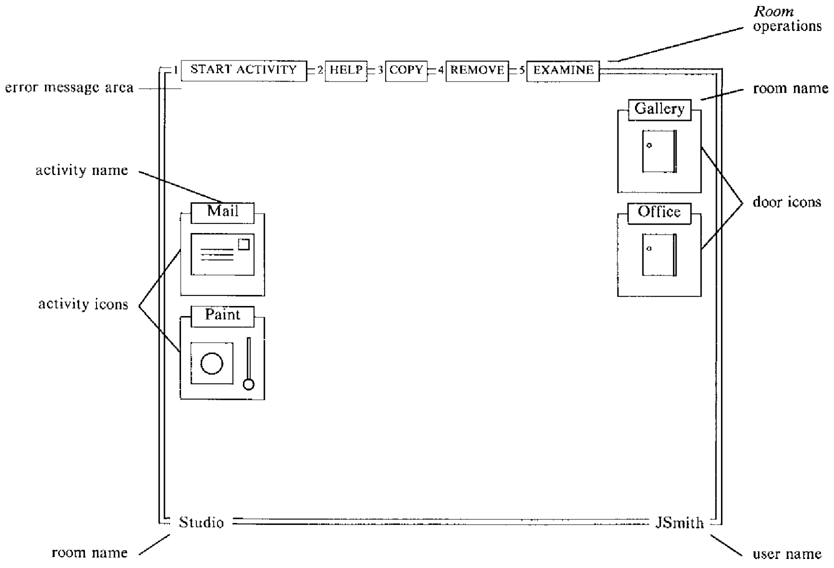


**Figure 3-1:** The *Room* Display

## 3.2 Room Operations

*Room* operations are invoked by first selecting an icon and then selecting an operation. There are two ways to select an operation. The first is to point to and SELECT it on the display. The second is to use a FUNCTION KEY on the keyboard. A FUNCTION KEY is associated with each operation. The numbering of the operations corresponds to the numbering of the FUNCTION KEYs so that pressing F3, for example, starts the COPY operation in *Room* (see Figure 3-1). An operation "blinks" when selected, indicating that the computer has received the command and that the operation is underway. *Room* supports five operations:

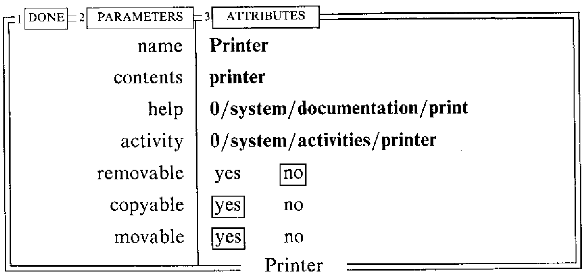
1. START ACTIVITY/ENTER ROOM – If the selected icon is a door icon, the operation reads ENTER ROOM; if selected, you move into the room to which the door leads. If the selected icon is an activity icon, the operation reads START ACTIVITY; if selected, the activity represented by the icon is started.

2. HELP – The HELP operation starts an activity which describes the selected icon. If the selected icon is an activity icon, the Help activity explains how to use the activity. If the selected icon is a door icon, the Help activity explains where it leads.

3. COPY The COPY operation replicates the selected icon if there is enough space in the room. This operation is useful for distributing icons to other rooms. For example, copying door icons and moving them to other rooms enables direct travel between any two rooms.
4. REMOVE The REMOVE operation erases the selected icon.
5. EXAMINE An icon has properties that determine how the icon behaves within *Room* and how its activity behaves. Selecting the EXAMINE operation allows you to examine the selected icon and alter any of these properties. Once altered, these properties remain unchanged until they are explicitly changed again. In general, an icon has two types of properties: *attributes* and *parameters.* All icons have attributes, but not all icons have parameters.

## 3.3 Icon Attributes

An icon's *attributes* determine how it behaves within *Room* and *what* activity is started by the icon. Figure 3-2 shows the attributes of an icon.



| | |
|---|---|
| name | **Printer** |
| contents | **printer** |
| help | **0/system/documentation/print** |
| activity | **0/system/activities/printer** |
| removable | yes [✓ no] |
| copyable | [✓ yes] no |
| movable | [✓ yes] no |

Printer

**Figure 3-2:** Attributes of an Icon

Every icon has seven attributes associated with it.

- *name–* specifies what name appears on the icon. Any name which does not fit on an icon is truncated when displayed.
- *contents–* specifies what picture appears within the icon's borders. *Room* has a predefined set of icon pictures. A partial list follows.

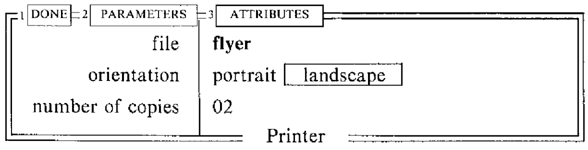
| | | | |
|---|---|---|---|
| calculator | clock | document | door |
| floppy | hanoi | hardisk | life |
| mail | network | paint | garbage |
| terminal | graft | printer | ledger |

If one of these names is specified as the contents attribute, a picture is displayed. Otherwise, the text in this field appears within the displayed icon.

- *help*– specifies the pathname of a file containing instructional information about the icon. The Help activity is given this pathname when the HELP operation is invoked.
- *activity*– specifies the filename of the activity that is to be invoked when the icon is started.
- *removable*– allows an icon to be removed using the REMOVE operation.
- *copyable*– allows an icon to be copied using the COPY operation.
- *movable*– allows an icon to be moved about in the room or to other rooms.

## 3.4 Icon Parameters

An icon's *parameters* determine *how* an activity behaves. Selecting the PARAMETERS operation displays an icon's parameters, if any. Figure 3-3 shows the parameters of a printer icon; the printer icon has three parameters.



| | |
|---|---|
| file | **flyer** |
| orientation | portrait [landscape] |
| number of copies | 02 |

**Figure 3-3:** Parameters of an Icon

The *name* of the parameter provides a brief description of the parameter. For example, any number of copies (from 1 to 99) of the file "flyer" can be printed by altering the *number of copies* parameter. Moreover, the copies can be produced in either of two orientations by altering the *orientation* parameter.

## 3.5 Modifying Parameter and Attribute Fields

There are three field types for describing parameters and attributes; each field type is manipulated differently.

1. *Options field* – displays a selection of possible values for a parameter or attribute. The *orientation* parameter in Figure 3-3 is an example of an options field that has two options – *portrait* and *landscape*.

   Only one of the displayed options can be in effect at any time; the selected option is highlighted.
2. *Numeric field* – has a numerical value. This type of field appears as one or more digits. The *number of copies* parameter in Figure 3-3 is an example of a numeric field. Each digit in a numeric field behaves like a thumbwheel; selecting a digit increments it by one.

3. *Text field* – must be filled by typing; this field is used whenever it is impossible to use a number or a selection of values. Screen editing features are available in text fields.

## 3.6 A World Based on Rooms

The discussion so far has dealt with operations that are available in *Room*. We have not discussed the environment in which *Room* is used. This environment solves problems which are not addressed by *Room*: How do users share a workstation? What about security? Where does one acquire icons? How are icons distributed? This section develops the room metaphor as an environment to address these problems.

### 3.6.1 The Office

Every user authorized to use a workstation has a personal office. As a new user to the system, you are provided with an office with some general icons. Your office is an essential room as it is the only entry point to your portion of the building.

### 3.6.2 The Lobby and Entrance

When a workstation is switched on, you find yourself in the Lobby. The Lobby contains a door labeled *Entrance* that leads to your Office. Similarly, in your Office there is a door labeled *Exit* which leads back to the Lobby. When you have finished working, you should always return to the Lobby. This voids your privileges on the system and protects your resources from other users of the workstation.

Unlike most doors, the Entrance does not immediately lead to another room; instead, it establishes a security barrier which prevents unauthorized access to the workstation. It requires authorized users to identify themselves to the system because different users have different privileges and individually tailored environments. The Entrance displays a list of names of users who are authorized to use the system. You must select your user name and type your password on the keyboard. If your password is accepted, you will enter your Office; otherwise, you will return to the Lobby. The Entrance also allows you to change your password. You must first provide your current password before the new password is accepted.

### 3.6.3 The Supply Room

The Supply Room provides a facility for acquiring new icons. Since the Supply Room is shared by everyone, the icons within it can be neither examined nor repositioned. The desired icons must first be copied and then moved out of the Supply room via the Exit. In the Supply Room, the door labeled *Exit* leads back to the room from which you entered the Supply Room, usually your Office.

### 3.6.4 The Truck

A Truck is a room residing on a floppy diskette. The floppy disk must be in the diskette drive before the Truck can be entered. As with the Supply Room, you can return to the room from which the Truck was entered via the Exit.

Trucks are used for transporting icons among workstations. Trucks are used to distribute new software and to perform updates to existing software. In a Truck, there may be icons for copying files from the floppy diskette onto a hard disk or a file server.

### 3.6.5 The Room Maker

The Room Maker is an activity that constructs various kinds of rooms. It is started using a "make room" icon. The Room Maker has blueprints of rooms for many of the tasks that Port supports; there is a Port program development room, a document creation room, an arcade, and even an empty store room. The Room Maker requires the name of the new room and its type. It then prompts for details necessary for the construction of the specified type of room. An empty store room, for example, requires no further questions; but a document creation room requires the name and location of the document and the document type (memo, thesis, paper, help file, etc.). With this information, the Room Maker builds a custom room which is tailored specifically to the task.

With a general purpose print icon, for example, at least two steps are required to print a document, first finding the document and then sending it to the printer. However, because the Room Maker knows which document you are working on, it can create an icon which starts an activity to print the document. In short, a Room Maker not only finds the right tools for a given task, but also adapts them to provide more efficient interaction.

# 4

# Learning Considerations in User Interface Design

What methods can the designer employ to accelerate a novice's understanding of his system? This chapter examines the theoretical models of mental processes which occur during learning. Our goal is not only to find ways to ease the mental effort that accompanies learning, but also to capitalize on this knowledge, once learned.

The format of our presentation is as follows. First, we briefly describe a well-studied learning phenomenon†, concentrating on those aspects that either deter or promote learning. Then, following each discussion, we present several general user interface design principles that either encourage or discourage the conditions which promote or deter learning, respectively. Each principle is accompanied by design strategies that prescribe a number of ways in which the principle can be exploited. These strategies are illustrated with examples of their usage by the systems we have described in previous chapters.

## 4.1 Information Acquisition

Learning considerations are divided into two categories: *information acquisition* and *information transfer.* Acquisition refers to the processes and events that occur as knowledge is encoded and stored into memory. Transfer refers to the effects of prior experiences on acquiring information.

### 4.1.1 The S-R Contiguity Principle

A prominent theory in the psychology of learning states that learning occurs by the formation of primitive associations between *stimuli* and *responses.* The strength of an *S-R association,* as it is called, is revealed by the probability of a response occurring in a stimulus situation. The strength of an S-R association is influenced by the outcome of the response or *reinforcement,* which may be positive or negative. *Positive reinforcement* strengthens the bond while *negative reinforcement* weakens the bond. There are active and passive aspects of the theory.

- **Classical Conditioning.** In the passive aspect, called *classical conditioning,* the subject is conditioned to react to a stimulus which does not

† The material on learning psychology is taken from several introductory texts, [Houston 1981, Marx and Bunch 1977, Price et al. 1982, Tarpy and Mayer 1978].

normally elicit the desired response. This is done by pairing the *conditioned stimulus* (CS) with an *unconditioned stimulus* (UCS) that unconditionally causes the reaction. Soon the UCS can be removed with the result that only the CS is needed to cause the response. There are various factors that affect the permanence of this association which we do not detail here. For example, by withholding the UCS, the conditioned response will eventually be extinguished.

• **Operant Conditioning.** The active aspect of S-R association theory, called *operant conditioning*, is more applicable to the study of the novice user. Unlike classical conditioning, operant conditioning does not directly involve a stimulus that unconditionally elicits a response; the subject may or may not respond when presented with the stimulus. Learning comes by responding (or not responding) to a stimulus and noting the effects. If there is reinforcement, a bond will form between the stimulus and the response.

Many learning theorists believe that learning will not occur if the response and reinforcement are not *temporally contiguous* – that is, they must occur very close in time. The necessity of temporal contiguity has not been proven, although it is one of the oldest assumptions in the field. Most demonstrations of non-contiguous learning have been countered by arguments asserting that memory was the medium that bridged time, providing the necessary temporal contiguity. These arguments have been hard to disspell.

There is much evidence indicating that temporal contiguity is not sufficient for learning to occur. Some experiments demonstrate varying degrees of learning when factors such as reinforcement, practice, intent, motivation, and the type of stimulus are modified. For example, if the CS occurs with or without the UCS with equal probability, then learning does not occur. Although the results indicate that these factors must be taken into account, they lie beyond the scope of our discussion.

#### 4.1.1.1 Immediate Feedback

For our purposes, it is enough to simply accept the indisputable influence of temporal continuity on learning. This phenomenon clearly translates into one of the most frequently cited design principles:

*Reinforcement should occur very soon after a user's action.*

This principle does not suggest that applications must be made as fast as the user is able to respond; inherent computational complexities may make this impossible. Rather, stimulus should be given simply to show the user that his actions have been accepted or ignored. For example, some systems with graphics capabilities change their cursor's shape into an hourglass or bee to signify busyness.

Besides deterring learning, lack of immediate feedback is responsible for other adverse effects. Norman (1983) attributes the lack of adequate feedback to *mode errors* which occur when the user presumes to be in one state but is actually in another. Actions are made out of context and hence have unknown and possibly disastrous effects. For example, in a heavily loaded system that buffers input without adequate feedback, a user may retry an operation because he is unsure of whether or not it was accepted the first time. If the operation deletes a line, one can imagine the user's dismay when his text slowly disappears when his "unintentional" operations are finally executed.

### 4.1.2 Concept Learning

A concept represents a group of objects that share common properties. A concept is characterized by a set of *attributes* and *rules* that relate the attributes. Hence, concept learning involves two stages of learning: first, there is *attribute learning* where the learner must identify the relevant dimensions of the stimulus; then there is *rule learning* where the learner must deduce the appropriate rule which relates the attributes.

There are two popular theories that attempt to describe the process of acquiring concepts.

- **The Continuity Theory** views concept learning as an extension of the S-R model of learning. In this theory, a concept is learned by tallying the trials that result in positive reinforcement. After a few trials, the learner will have accrued a mental table of strong possibilities for the relevant attributes and rules of the concept. In this way, a concept is gradually acquired.
- **The Noncontinuity Theory** views concept learning as a process of inducing and testing hypotheses. The experimenting learner constructs a hypothesis and adheres to it until it fails. Learners can adopt different hypothesis testing strategies for different learning situations. Learning, in this theory, comes in discrete steps.

The results of experiments suggest that concept learning involves both theories. However, adults and verbal children tend to hypothesize rather than form rote associations. For the remainder of this section we concentrate on the hypothesis testing model of concept formation.

There are many factors that influence a concept learning effort. For example, attributes are known to have different discriminable levels. A concept with salient attributes is easier to learn than one comprised of nondescript characteristics. Similarly, there are rules which are easier to deduce than others. For example, it has been observed that a conjunctive concept takes less time to master that a disjunctive concept. A conjunctive concept is one where the correct hypothesis is a conjunction of features (e.g. red *and* box). Similarly, a disjunctive concept is one where the correct hypothesis is a disjunction of features (e.g. green *or* round). For our discussion, we concentrate on a less involved property of the hypothesis testing model – simply, the fewer hypotheses there are to try, the faster a concept can be learned.

#### 4.1.2.1 Trial Space Reduction

In both attribute and rule learning, the acquisition time can be decreased by reducing the number of dimensions or eliminating irrelevant dimensions of the stimulus. By restricting the pool of hypotheses or trial space, there is less chance the learner will choose an incorrect hypothesis and waste time testing it.

*The pool of hypotheses should be reduced as much as possible.*

There are numerous ways to trim the trial space. We shall mention only a few general strategies.

### Trial Space Reduction by Immediate Negative Feedback

We have discussed the necessity of immediate feedback for learning. But simply acknowledging every action that is made yields a space of possible actions too large to explore in a reasonable amount of time. Not every action is meaningful; it depends on preceding actions. A mechanism is needed to restrict the trial space, permitting meaningful actions and disallowing others.

*Negative reinforcement should occur soon after an incorrect action.*

Thus actions must be interpreted as well as acknowledged. For example, to make a file in Unix, the user types the appropriate command followed by the file name. Not until he presses RETURN does he discover whether the file name is valid. By experimenting, the user would need many trials (and possibly many file removals) to determine, for example, the maximum length of a file name. The space of possible file names is large. In Port, however, the "make file" activity does not accept a character which is not allowed in a file name or which extends the file name past its maximum length. By pressing different keys the user can quickly discover the legal characters and the maximum length of a file name. The space of possible file names is greatly reduced. When the user finally makes the file (by selecting the MAKE FILE operation), the file name is guaranteed to be a legitimate one.

### Trial Space Reduction by Visibility

When one is confronted with an unfamiliar system, there are two levels of understanding which take place – syntactic and semantic information acquisition. Syntactic information refers to the names of available objects, operations and options and how to manipulate them. Semantic information refers to the effects or behaviour of these entities when they are used.

*Objects, operations and options should be made visible.*

The application of the visibility principle eliminates the need to experiment to determine the names of entities. In Unix, there is a barrier that prevents an experimenting novice from progressing past the syntactic acquisition stage. For example, to determine the options of a command, the user must try every letter of the alphabet. In such a case, consulting documentation is faster and less tedious. Port and Star have facilities (icon examination and property sheets, respectively) which display the available options. The user can simply try the various options and note their effects. It is typically faster to try an option rather than consult a manual.

However, the user should be made aware of which objects on the screen represent operations. Star combines the visibility and immediate negative feedback principles into a variant cursor whose shape changes depending on the type of object it is pointing to. Hence, by simply pointing to an object on the screen and noting the cursor's shape, one can determine if it is an operation.

Port also combines the negative feedback and visibility principles by indicating whether or not an operation is meaningful – meaningful operations are displayed in reverse-video while unmeaningful operations are displayed in bold. In this way, the user is informed whether an operation is applicable even before he tries it. For example, in the Examiner's window, if parameters are being examined, the PARAMETERS operation is bold and hence, cannot be selected.

**Trial Space Reduction by Organization**

Grouping together logical objects narrows the search space for a desired object. Moreover, it quickly excludes irrelevant objects in units of groups rather than on an individual basis.

*Similar objects should be grouped together.*

*Room*'s Room Maker observes this principle of organization by trimming the space of potentially useful tools for a given task.

## 4.2 Information Transfer

Information transfer is the influence of prior experiences on the acquisition of new material. Prior experiences may facilitate learning or they may hinder it. The transfer phenomenon makes it possible to exploit existing mental structures and considerably reduce the effort in learning similar concepts. Hence, transfer effects are of enormous importance in the design of user interfaces. We are interested in design strategies that promote positive transfer and mitigate negative transfer.

Transfer experiments involve two stages of learning. The experimental group must learn an *initial task* followed by the *transfer task.* The control group is either not taught an initial task or taught one that is entirely unrelated to the transfer task. The objective of such an experiment is to compare the performance of the two groups on learning the transfer task. There are three possible outcomes: if the experimental group performs better than the control, *positive transfer* has occurred; if their performance is worse, *negative transfer* has occurred. There is also the possibility of *zero transfer* where either the effects of prior learning are insignificant or the effects of negative and positive transfer cancel. For example, a user would experience negative transfer if the position of a familiar key on the keyboard were changed. A user learning an editor would experience positive transfer if he had used a similar one before; if he had never used an editor before, zero transfer should result..

Another distinction separates transfer effects into two other classes. *Specific transfer* refers to the influence of the specific information in the initial task on learning the information in the transfer task. *General transfer* refers to the effects of learning the task itself, such as the development of strategies, and not to the specific information provided by the task.

### 4.2.1 General Transfer

There are basically two kinds of general transfer.

- **Learning-to-learn.** As a subject is presented with tasks to learn, he develops strategies, habits and skills that subsequently help in learning similar tasks. This phenomenon is referred to as learning-to-learn and is characterized by the progressive improvement at performing a particular task.

Learning-to-learn transfer is relatively permanent. Studies show that retention of the specific information in the tasks degenerates, but the effects of learning-to-learn remain stable over test intervals. For example, if a user is versed in one or more programming languages, learning-to-learn transfer would help him learn another programming language.

• **Warm-up.** Warm-up is the physical and mental adjustments that accompany a switch in activities; such adjustments include attentional, sensory, postural and attitudinal changes. Warm-up effects are also characterized by an improvement in performance over time. However, warm-up transfer is less permanent; once the task ceases, the effects of warm-up rapidly dissipate. For example, warm-up transfer occurs as a user reads a program written in one language after having read programs written in another language. In order to understand the program, he must adjust to the differences between the two languages (i.e. programming rules, style and techniques).

General transfer is positive, provided learning proceeds in the same manner between tasks. If unaccounted for, a transfer experiment will yield either overestimated positive transfer or underestimated negative transfer. For example, if the control group were not taught an initial task, they would be at a slight disadvantage since the experimental group will benefit from general transfer by learning their initial task. Although its effects cannot be eliminated, they can be compensated for by having the control group learn an initial task which is entirely unrelated to the transfer task. Both groups then have equal opportunity to benefit from general transfer.

#### 4.2.1.1 The Null Application

The effects of learning-to-learn and the trial space reduction principle suggest that a learner's introduction to a system can be eased if he is first taught a *null application*.

*An application should be provided which demonstrates the interaction mechanisms of the system.*

A null application demonstrates all the important concepts that are necessary for interaction with the system. It should not necessarily perform anything useful because there should be minimal distraction understanding what the application does. This property observes the principle of trial space reduction. By the effects of learning-to-learn, the information acquired from interacting with the null application should transfer when confronted with a useful application.

In Unix, there is little need for a null application because the only interaction mechanism available is typing with the keyboard, and because there is little consistency concerning how applications are operated.

Although Port and Star may be ultimately easier to use than Unix, they may require a larger initial learning effort. In Unix, the user need only type a few characters to run his first command. In Port and Star, there are many preconditions which must be learned and understood before any meaningful interaction is possible. Concepts such as icon, window, active window, activity, activity name, selection, positioning, operations, hiding, growing, shrinking, etc. are needed for even the simplest applications. Port provides some simple games, such as a checkers activity, which serve as null applications.

### 4.2.2 Specific Transfer

A task in a transfer experiment is typically learning a list of paired words, called *paired-associate* lists. Performance on the transfer task is based on the ability to recall one term of the pair when presented with the other. A paired-associate list is symbolized as *X-Y* where X represents the stimuli (S) or cues and Y represents the responses (R). This notation is used to indicate the degree of similarity between the initial and transfer tasks. For example, a transfer experiment where the stimulus terms are identical and the response terms are unrelated, is known as the *A-B,A-C paradigm.*

It should be noted that transfer research has predominantly involved the learning of paired-associate lists. Some theorists feel that it is dangerous or impossible to generalize some of the findings of verbal transfer research to more complex task domains. Only those findings which we feel can be readily applied to designing user interfaces are discussed. In fact, computer interaction is currently closer to the methodology of verbal learning than many everyday activities since its mode of communication is primarily verbal.

Specific transfer involves at least four subprocesses. A prediction of overall transfer must consider the separate effects of each subprocess. The overall transfer is, in a sense, a sum of the various positive and negative influences of these subprocesses. Following the description of the subprocesses, we present various transfer paradigms that illustrate how the effects of these subprocesses affect the outcome of the resultant transfer.

1. **Response Learning** simply refers to the process of learning the response terms of a paired-associate list. It predicts positive transfer if some response terms of the initial list occur in the transfer list.

2. **Stimulus Differentiation,** like response learning, refers to the process of learning the stimulus terms of a paired-associate list. Additionally, in learning a list, subjects must also discriminate between the stimulus terms. The greater the similarity between the stimuli, the greater the difficulty in learning the list.

3. **Forward Associations** refers to the association of a response with a stimulus so that the response is elicited whenever the stimulus is presented. The effects of forward association predicts positive transfer if some S-R associations in the initial list occur in the transfer list.

4. **Backward Associations** can also form during paired-associate learning. Hence, negative transfer can occur if response terms common in both the initial and transfer list are associated with different stimuli.

## Transfer Paradigms

A subject can be made to experience either positive or negative transfer by varying such factors as the similarity and meaningfulness of the stimulus and response terms. The effects of the following basic transfer paradigms depend on how these factors affect each of the four subprocesses described above.

- **A-B,C-D Paradigm.** There is no similarity between either the stimulus or the response terms. This paradigm produces zero transfer. It is the basic control paradigm used to compensate for the effects of general transfer. Having the subject learn two completely different commands (e.g. print a file, send a mail message) is an example of an A-B,C-D paradigm.

- **A-B,C-B Paradigm.** The effects of response learning induces positive transfer but the effects of backward associations induces negative transfer. The overall effects of this paradigm depend on the *meaningfulness* of the responses. If the meaningfulness of the response terms is low, positive transfer results. This is because more time is spent in response learning, allowing less time for backward associations to form. Similarly, negative transfer results if the terms are highly meaningful. Changing the names of a familiar set of commands is an example of an A-B,C-B paradigm.

- **A-B,A-C Paradigm.** This paradigm causes negative transfer due to the formation of forward associations in learning A-B. The B terms compete with the C terms, interfering with the establishment of A-C associations. If the meaningfulness of the stimulus terms is low, then the positive transfer effects from stimulus learning can surpass the negative transfer effects from competing associations. This is because more time is spent in stimulus differentiation allowing less time for forward associations to form. An example of an A-B,A-C paradigm would be if the names of a familiar set of commands in one application were assigned to different commands in another application.

- **A-B,A-$B_r$ Paradigm.** In this paradigm, the stimuli and responses are identical but the response terms in the transfer list are *re-paired* (i.e. the associations between the stimulus and response terms are different). This paradigm causes massive negative transfer because the forward associations formed from the initial task interfere with the formation of new associations. The negative transfer effects can be mitigated if the terms are low in meaningfulness because more time is spent learning the terms than in forming strong forward associations. Changing the name-to-command assignments within a familiar set of commands is an example of an A-B,A-$B_r$ paradigm.

- **A-B,A-B′ Paradigm.** The stimuli in this paradigm are identical while the response terms in the initial list are similar to those in the transfer list. If the similarity is strong enough, positive transfer will occur from the effects of both forward associations and stimulus learning.

### 4.2.2.1 Consistency

Strong similarity between the responses in the initial and transfer tasks (the A-B,A-B' paradigm) promotes positive transfer. This fact readily translates into one of the more popular design strategies:

*Similar entities should behave in similar ways wherever they occur.*

This principle suggests, for example, that an operation such as deletion should be invoked by the same mechanism across all contexts whether it refers to a mail message, a line of text or an entire document.

Although consistency is desirable, it is not always easy to achieve. There are many (sometimes conflicting) dimensions along which a design decision can be consistent. Smith et al. (1982) relate one such design dilemma in Star – what happens to a document icon after it is printed? The icon can be deleted, returned to its initial location, or left in the printer. All three alternatives are consistent with different models. The first is consistent with the MOVE command. Whenever an icon is moved to the file cabinet, mailbox or garbage, it is deleted (after it is filed or mailed, of course). The third is consistent with a user's model of a physical printer. However, the second was chosen for various pragmatic reasons and also because it saves the user a step (removing the icon from the printer). Hence, other considerations may nullify the consistency principle.

**Consistency by Universal Operations**

*The design should strive toward a small set of powerful, universal operations.*

A powerful and universal operation applies in many or all applications in the system. For example, Star's MOVE command can be used to move text, move documents, print documents and mail documents; the PRINT and MAIL commands are thus absorbed by the MOVE command.

Another instance of employing universal operations is Port's editing facilities. Port makes available its basic text editing features in every application that requires text as input.

**Consistency by Hardware Transparency**

*The system should avoid separate interaction mechanisms because of hardware differences.*

Unix works consistently across a wide variety of terminal devices, irrespective of their sophistication. As a result, the user can easily switch between terminals with almost no learning cost. However, this consistency is maintained at the cost of wasting certain capabilities a terminal device may offer.

Port strives to conceal the presence of a network so that remote and local files are accessed in an identical manner. Hence, all applications which reference files operate in the same way, regardless of where the files are physically located.

### 4.2.2.2 Stimulus Differentiation

*Dissimilar entities should be made to appear differently.*

If similar stimuli are associated with different responses (the A-B,A-C paradigm), a conflict will arise as to which response should be elicited. The stimulus differentiation principle suggests that making dissimilar objects and operations appear differently will reduce or remove this potential source of interference.

## Consistency by Dimensional Expansion

The stimulus differentiation principle constrains the ways in which consistency can be applied. Often, a "special case" is forced into a consistent mold and the resulting cracks hidden for the sake of maintaining consistency. This practice may make the concept easier to learn but only postpones the confusion that inevitably follows.

> *At the first sign of an inconsistency, the dimensions of consistency should be expanded to accommodate the inconsistency.*

In Unix, all devices are generalized as files. Some devices, such as teletypes, card readers and printers, can be considered restrictions of the file concept and hence, are compatible. However, devices such as modems, tape drives, typesetters, graphic terminals and network connectors do not fit well into the file concept. As a result, there are many "special files", each marring the concept in its own idiosyncratic manner.

*Room* supports a higher level abstraction, icons, which does not disguise the dissimilarities of devices as files. Rather than delude the learner with a false sense of simplicity, *Room* addresses the problems of helping him assimilate the multitude of available services. Although consistency between devices is lost, consistency is maintained along more appropriate dimensions such as the method of using icons and an icon's similarity to its physical counterpart.

## Modes and Escaping

Besides restoring consistency, dimensional expansion is one of the most powerful techniques available to the designer in simplifying a user interface. For example, its application can eliminate *modes* and the need for *escaping*. A mode is a state that prevents certain other useful operations from being invoked unless the state is left. Escaping is used to create new objects from a finite collection of objects. Unix's *tip* illustrates an application which uses escaping to represent special operations. Tip enables a user to connect to another Unix host computer without leaving the current session. When invoked, it directs all input from the terminal to the new session instead of the old session. The problem is how to disengage this redirection of input and regain communication with the old session. A command cannot be constructed from the normal character set as any character and character sequence can potentially be used in the new session. This is solved by choosing a character from the normal character set and endowing it with the properties of an *escape character*. The escape character and the character(s) following it, are interpreted as a special operation by the application. Two occurrences of the escape character are interpreted by the application as a single normal instance of that character.

There are basically two ways to achieve dimensional expansion: "display partitioning" and the creation of new objects. In display partitioning, the display is divided into distinct regions. Keyboard input is interpreted differently depending on which region contains the cursor. For example, windows partition the display between applications. Windows enable a user to interact with other applications without quitting the current one; hence, windows eliminate a mode. Port's operations further partition the display within a window.

The user interface of the tip application can be simplified by providing new keys on the keyboard and have tip recognize them. Alternatively, the display can be partitioned into two regions; keyboard input in one region is sent to the host, while keyboard input in the other is interpreted by tip.

*Room* adds a dimension to the desktop concept by essentially providing "multiple desktops" in the form of multiple rooms. This is made possible by providing a special icon that Star lacks – the door.

#### 4.2.2.3 Analogy

Using analogies or metaphors in the design of a user interface is also a way of exploiting transfer. Like consistency, it takes advantage of existing mental structures; a learner can draw upon the analogy to help him understand an unfamiliar situation.

*Designing around a familiar metaphor helps reduce learning time.*

However, there is an important difference between consistency and analogy that makes the use of analogies potentially detrimental. Consistency makes use of "known" experiences and can therefore be predicted and controlled. Analogy appeals to experiences which are not fully known; the effectiveness of an analogy relies on the extent to which the users's analogical model coincides with that of the designer's. If either the user or the designer misinterprets the analogy (interprets too many, too little or the wrong aspects of the analogy), incorrect associations may result in negative transfer.

**Bounded Analogies**

A useful analogy that demonstrates how analogies can be misused is the *adage*. An adage such as "Getting things done around here is like mating elephants" does little to enlighten the learner about how things are actually done. Not until the punch line is delivered can the learner even begin to suspect the similarities between the two models. In this case there are three – (1) it's done at a high level, (2) it's accomplished with a great deal of roaring and screaming, and (3) it takes two years to produce results.

Halasz and Moran (1982) suggest that an analogy should be used as a *literary metaphor* and not as a model for reasoning about a computer system. They feel that only an abstract conceptual model that accurately represents the system should be used for detailed reasoning about the system. The role of the literary metaphor is to impart to the learner a specific characteristic of the conceptual model – that is, to help construct the conceptual model. In other words, an adage like the one above, is an effective way to build a model but should not be used as the model itself. By clearly indicating what aspect of an analogy is relevant, this suggestion helps to bound the analogy.

> *Explanations of the limits and incompatibilities of an analogy should accompany its use.*

This principle restricts the space of misconceptions which may arise if the user is not informed of the limits of an analogy. For example, a useful bounded analogy for doors in *Room* would be "As in real life, doors are used to enter other rooms; however, in *Room*, doors can be copied and moved about".

## 4.3 Summary

We have accounted several precepts from learning psychology and inferred from each, a number of user interface design principles. The work of this chapter is a first approximation of the design principles framework. Even in this primitive form, the framework reveals relationships between popular design principles. For example, our categorization relates three of Xerox's design principles (described in Chapter 2). Universal commands is a *strategy* for achieving the consistency principle; the consistency and familiar user's model principles are both ways to promote specific transfer. However, there are several deficiencies with our framework, the most prominent being a lack of any quantitative analysis. For example, although our visibility strategy helps the trial space reduction principle, it creates another problem, namely, searching for displayed objects; the search time is directly proportional to the number of displayed objects. Because there is a tradeoff with using this strategy, it is necessary to have a quantitative relation that allows the designer to determine whether visibility should be applied in a particular circumstance. Development of a more quantitative approach is a necessary step in the development of the framework.

## 4.4 References

[Houston 1981]
J.P. Houston, *Fundamentals of Learning and Memory 2E*, New York: Academic Press Inc., 1981.

[Malasz and Moran 1982]
F. Halasz and T.P. Moran, "Analogy Considered Harmful", *Proceedings of the Human Factors in Computer Systems Conference*, Gaithersburg, MD, March 15-17, 1982.

[Marx and Bunch 1977]
M.H. Marx and M.E. Bunch, *Fundamentals and Applications of Learning*, New York:Macmillian Publishing Co., Inc., 1977.

[Norman 1983]
D.A. Norman, "Design Rules Based on Analyses of Human Error", *Communications of the ACM*, Vol. 26, No. 4 (April 1983), pp. 254-258.

[Price et al. 1982]
R.H. Price, M.Glickstein, D.L. Horton and R.H. Bailey, *Principles of Psychology,* Holt, Rinehart and Winston, The Dryden Press, Saunders College Publishing, 1982.

[Tarpy and Mayer 1978]
R.M. Tarpy and R.E. Mayer, *Foundations of Learning and Memory,* Illinois:Scott, Foresman and Company, 1978.

# 5

# Anatomy of the *Room* Environment

The *Room* environment is implemented as a set of cooperating processes. This separation into processes reduces the average amount of main memory used by the *Room* environment and enhances the responsiveness of *Room*'s user interface. Figure 5-1 depicts the genealogical and communicative relationships of the various *Room* environment processes.

With the exception of the Room Makers, this chapter presents the design of the *Room* implementation under Port Version 2.0. The Room Maker described here is as yet in a prototypical stage of development. We begin with a brief description of the Port process model and then discuss *Room*'s major data structures and the role of each process.

## 5.1 The Port Process Model

Port is a multi-process operating system where process communication and synchronization is achieved with four message-passing primitives [Cheriton et al. 1979, Cheriton 1982, Malcolm et al. 1983].

```
send( message, reply_message,receiver_pid)
receive( message, sender_pid)
sender_pid= receive_any(message )
reply( reply_message,sender_pid)
```

Each process is identified by a unique process identifier (PID). The send primitive is used to send a message to, and await a reply from the process specified by receiver_pid. The receiver process can receive the message with either the receive or receive_any primitive. With the receive primitive, a particular process can be specified from which a message is expected. If no message is immediately available from that process, the receiving process waits. Other sending processes are suspended until the receiving process is ready to receive its messages. With the receive_any primitive, a message from any process is accepted. After processing a request, the receiver process can execute a reply primitive to return a reply message to the sender process. The sender process resumes execution once it obtains the reply message. Figure 5-2 illustrates the send/receive/reply interaction.

Dynamic process creation and destruction are done with two process management primitives.

```
pid = create( program_name,priority )
destroy( pid )
```

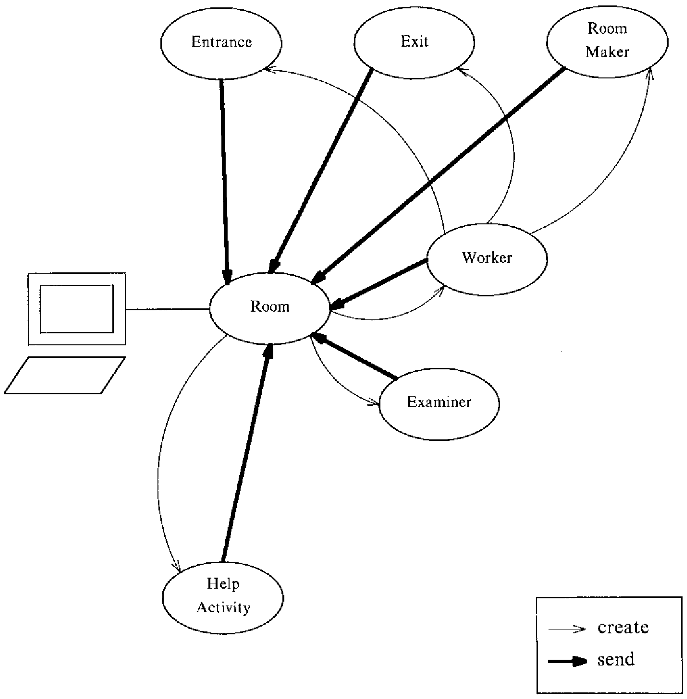


**Figure 5-1:** Process Structure of the *Room* Environment

The create primitive creates a new process at the specified priority and returns its PID. The destroy primitive terminates a process' execution. The operating system reclaims the resources used by the destroyed process and releases any processes that are awaiting a message from the process.

## 5.2 Data Structures and Files of the *Room* Environment

### 5.2.1 The Icon Descriptor

The Icon Descriptor is a record that contains information about an icon. This data structure is used by all but the Exit and Entrance processes. There are eight fields in an Icon Descriptor (see Figure 5-6). The LABEL, CONTENTS, HELP_FILE and ACTIVITY fields are pointers to strings containing the icon's name, contents,

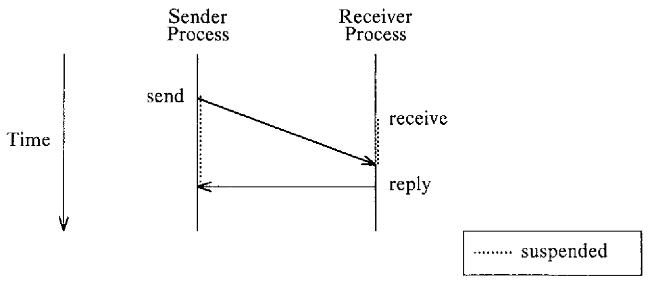


**Figure 5-2:** The send/receive/reply Interaction

help and activity attributes, respectively. The values of the remaining attributes – removable, movable, copyable and examinable – are stored in the ATTRIBUTES field. Since the values of these four attributes are either true or false, each is represented by a single bit in the ATTRIBUTES field. The icon's parameters are stored in the PARAMETERS field. If the contents attribute specifies an icon picture, the PICTURE field points to a string describing the icon picture. The INDEX field contains the icon's position in the room.

Icon descriptors are frequently sent between *Room* processes. The *Room* processes use two functions, Send_icon and Receive_icon, to exchange Icon Descriptors. These functions send the descriptor and associated strings with a series of sends, receives and replies.

### 5.2.2 The Users Tree

Users' files are stored in the *Users Tree.* Directly underneath the root of the Users Tree are files with the names of the users authorized to use the workstation. Figure 5-3a shows the Users Tree with two user subtrees, JPPChan and JSmith. A new user is added to the system by creating a user subtree in the Users Tree. Similarly, a user's privileges can be revoked by removing his user subtree from the Users Tree.

### 5.2.3 Room Files and the Rooms Subtree

Icon Descriptor information is stored in *room files.* There is one room file for each room. The name of a room file is that of the room it represents. A user's room files are stored in the Rooms Subtree located in the user's subtree (see Figure 5-3a). Public rooms, such as the Supply Room and Lobby, are stored in the Rooms Subtree located in the *System Tree* (see Figure 5-3b).

There are 29 lines of text in a room file. Figure 5-4a shows the room file of JSmith's Studio (refer to Figure 3-1). The first line contains a single character that represents the format version of the room file. This version indicator prevents

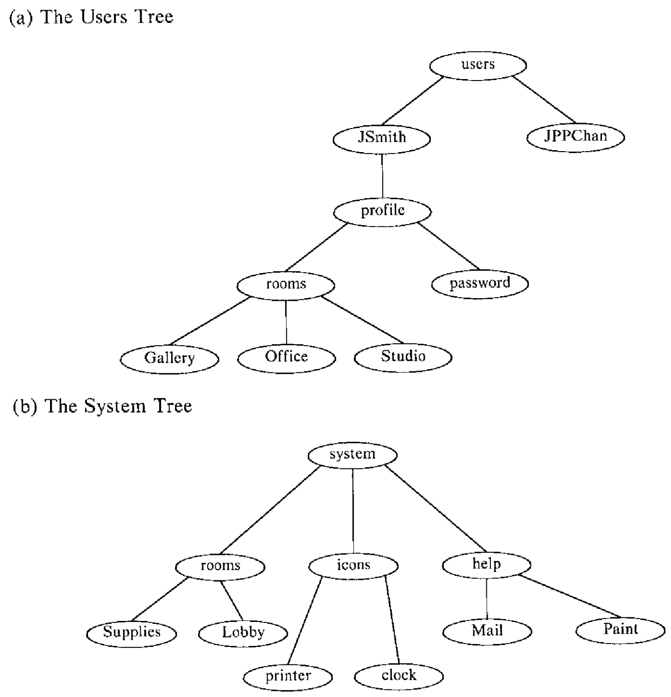


**Figure 5-3:** The Users and System Trees

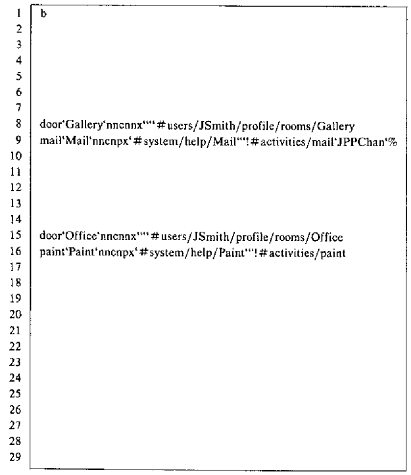
*Room* from misinterpreting old room file formats.

Each of the remaining 28 lines holds information contained in an Icon Descriptor. A line is constructed by concatenating the values in an Icon Descriptor and separating each value by a delimiter character (‘). Each of the 28 lines corresponds to an icon position in the *Room* display; vacant icon positions are represented by empty lines. Figure 5-4b shows the format used to store Icon Descriptor information in a room file.

### 5.2.4 Icon Pictures and the Icons Subtree

Each *icon picture* is represented by a string of bytes which, when displayed, renders an appropriate graphical object. Each icon picture is stored in a separate file. A collection of icon picture files is stored in the Icon Subtree located in the System Tree (see Figure 5-3b). An icon picture is identified by the filename in the

(a)



```
b






door‘Gallery‘nncnnx‘“‘#users/JSmith/profile/rooms/Gallery
mail‘Mail‘nncnpx‘#system/help/Mail“‘!#activities/mail‘JPPChan‘%





door‘Office‘nncnnx‘“‘#users/JSmith/profile/rooms/Office
paint‘Paint‘nncnpx‘#system/help/Paint“‘!#activities/paint













```

(b)

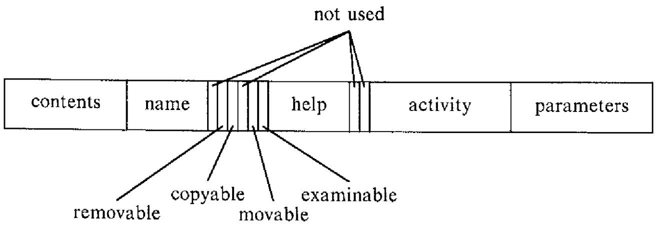


**Figure 5-4:** The Room File Contents for JSmith's Studio

icon's contents attribute. The Icons Subtree in Figure 5-3b has two icon picture files – a clock and a printer.

An icon picture is created using the Port Editor. A file containing a tablet of special characters is provided in the Icons Subtree for composing icon pictures, as it is inconvenient to generate these special characters using the keyboard.

### 5.2.5 The .parameters File

An Icon Descriptor contains only the values of an icon's parameters. The Examiner acquires the names and types of the icon's parameters from the *.parameters file*. All icons with parameters have a parameters file beneath the program file specified by the icon's activity attribute.

```
3
file
t
orientation
o [portrait] [landscape]
number of copies
n
```

**Figure 5-5:** The .parameters File for Figure 3-3

Figure 5-5 illustrates the .parameters file for the printer icon of Figure 3-3. The first line in a .parameters file contains the number of parameters. This enables the Examiner to pre-allocate enough space for all the parameter fields. A parameter field occupies two lines; one line defines the field name while the other defines its type. Type 't' specifies that *file* is a text field. Type 'o' is followed by the options available for the *orientation* parameter. Type 'n' specifies that *number of copies* is a numeric field.

## 5.3 The Room Process

Of all the processes comprising the *Room* environment, only the Room process is always executing; all other processes are created by the Room process when needed, and destroyed when no longer needed. In addition to creating helper processes, the Room process interprets user input, manages the *Room* window, and maintains various data structures. File I/O is done by the Worker to save space in the Room process.

### 5.3.1 The Room Descriptor and Room List

As Icon Descriptor information is read in from a room file, it is saved in a Room Descriptor. Figure 5-6 shows a Room Descriptor. The ROOM NAME field is a pointer to the room's name. The SLOTS field is a pointer to an array of Icon Descriptors. A room's Icon Descriptors are stored in the SLOTS array. This array has 28 entries, one for each icon in the room. The ROOM LINK field is used to chain Room Descriptors into a linked list.

Room Descriptors are stored in the Room List in main memory to make traveling between rooms very fast. However, to prevent a room file from becoming inconsistent with the Room Descriptor in memory, the Room Descriptor information is written into the room file whenever the user changes the room. Copying, removing and repositioning an icon causes the Room process to update the room file. An icon examination that results in a change to the icon's parameters or attributes also causes the room file to be updated.

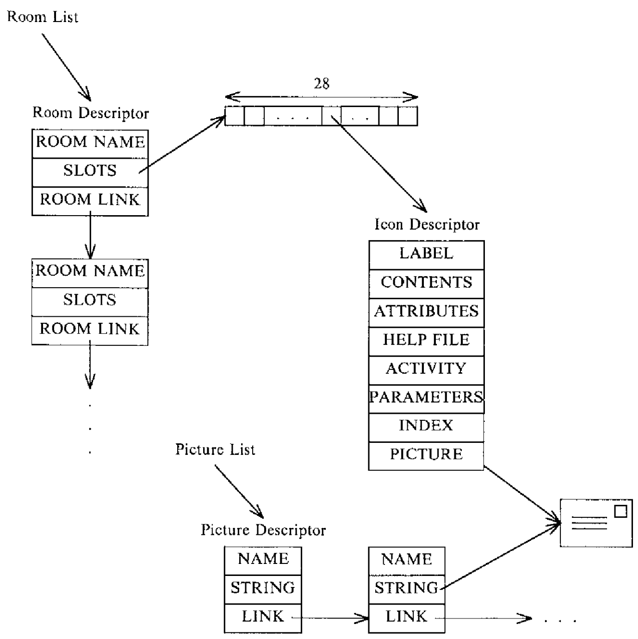


**Figure 5-6:** Data Structures in the Room Process

The Room List is a linked list of Room Descriptors.

### 5.3.2 The Picture List

An icon picture is stored in a Picture Descriptor (see Figure 5-6). The NAME field is a pointer to the picture's name while the STRING field is a pointer to a sequence of bytes that represents the picture. The LINK field is used to chain Picture Descriptors into a linked list, allowing the list of pictures to grow with time.

When an icon references an icon picture which is not in the Picture List, a Worker is created (described below) to obtain the picture. In this way, icon pictures are stored in main memory as needed.

### 5.3.3 The Room Stack

The Room process maintains a *Room Stack* to record the user's excursions between rooms. Before *Room* enters another room, the Room process pushes the name of the current room onto the Room Stack. Thus, the return route is available by popping the Room Stack. The main motivation for having a Room Stack instead of direct room connections is to permit several users to share a single room (e.g. the Supply Room). Since a shared room can be entered from any room, it is impossible to construct doors leading back to every room having direct access to the shared room. The Room Stack enables the return route to be determined dynamically.

The Room Stack is implemented as an array with currently 10 entries. The array structure limits the "depth" of rooms a user can be from his Office, and thus limiting any cycles that may occur.

### 5.3.4 Room Process Requests

In addition to handling user requests, the Room process also handles requests from other processes.

- ADD NEW ICON is a request used by applications that wish to place a new icon in the current room. The Room Maker uses this request to display a new door. If the current room cannot accommodate a new icon, the Room process replies with an appropriate failure message.

- ENTER ROOM forces the Room process to leave the current room and enter the specified room. The Exit and Entrance processes, for example, make use of this request to enter the Lobby and Office, respectively.

  There are three types of ENTER ROOM requests. The PUSH CURRENT type makes the Room process push the current room onto the Room Stack before entering the specified room. The FORGET ALL type makes the Room process clear the Room List and Room Stack before entering the specified room. The FORGET CURRENT AND POP type makes the Room process free the Room Descriptor of the current room, enter the room specified by the top of the Room Stack and finally, pop the Room Stack.

- DISPLAY MESSAGE is a request to display a message in the error message area (see Figure 3-1).

## 5.4 The Examiner

The Examiner is created by the Room process when the user selects EXAMINE. The primary reason for separating the icon examination function in a process is to economize on the amount of main memory used by *Room*. The Room process relinquishes control of the display and redirects all keyboard input to the Examiner. The Examiner obtains the Icon Descriptor of the selected icon from the Room process and displays the icon's attributes and parameters for manipulation by the user. When the examination is completed, the modified properties are sent back to the Room process and the Examiner terminates. The Room process then regains control of both the input stream and the display.

### 5.4.1 The Field Descriptor

In the Examiner, each icon parameter and attribute is stored in a *Field Descriptor* (see Figure 5-7).

| Table Descriptor |
|---|
| NUMBER |
| MAX_NAME_SIZE |
| LIST |

| Field Descriptor |
|---|
| NAME |
| VALUE |
| TYPE |
| OPTIONS |

**Figure 5-7:** Table and Field Descriptors

The NAME field is a pointer to the parameter's or attribute's field name. The TYPE field holds the parameter or attribute's field type. If the TYPE field contains an 'o', the VALUE field points to a list of options. The OPTIONS field specifies which option in the list of options is selected. If the TYPE field contains a 't' (text) or 'n' (numeric), the VALUE field points to a string containing the value.

Field Descriptors are stored in tables. The Examiner maintains two tables, one for the icon's parameters and the other for the icon's attributes. A table is represented by a *Table Descriptor* (see Figure 5-7). The LIST field is a pointer to an array of Field Descriptors. The NUMBER field holds the number of Field Descriptors in LIST. The MAX NAME SIZE field holds the maximum length of the field names. This value is used to minimize the width of the field name display area in the Examiner's window (see Figure 3-3).

## 5.5 The Worker Process

The Worker process is *Room's* interface to the file system; it stores Icon Descriptor information in room files, reads icon pictures, and starts activities. The Worker is a transient process created by the Room process only when needed to service a request. The Worker destroys itself after each request (to save space).

### 5.5.1 Worker Process Requests

The Worker handles four types of requests. All of the Worker requests are sent by the Room process.

- READ ICONS FROM ROOM is a request to obtain Icon Descriptor information from the specified room file. The Worker checks the version indicator to ensure that the room file format is current. If the format is out-of-date, the Worker destroys itself. Each line of the room file is read and packaged into an Icon Descriptor which is then sent to the Room process.

- WRITE ICONS TO ROOM is a request to save Icon Descriptor information in the specified room file. The Worker first writes the current version indicator into the room file. Then for each Icon Descriptor, the Worker writes a line of text representing the contents of the Icon Descriptor into the room file.

• READ ICON PICTURE is a request to locate and obtain an icon picture from *Room's* collection of icon picture files. The Worker searches for a file with the given name in the Icons Subtree (see Figure 5-3b. If the search is successful, the picture is returned.

• EXECUTE A PROGRAM is a request to start and pass parameters to an activity.

## 5.6 The Help Activity

The Room process creates a Help activity in response to a HELP operation. After receiving the Icon Descriptor of the selected icon from the Room process, the Help activity attempts to read the help file specified by the help attribute. If the read succeeds, the Help activity creates its own window and displays the contents of the help file. Otherwise, the Help activity sends a DISPLAY MESSAGE request to the Room process to display an appropriate message.

Help files may be located anywhere; by convention, help files for system icons are stored in the *Help Subtree* located in the System Tree. Figure 5-3b shows help files for the Mail and Paint applications.

## 5.7 The Room Maker Activity

The Room Maker icon starts the Room Maker activity. The Room Maker creates a window and prompts the user for the new room's name and type. The Room Maker checks if the room name is already in use. If so, it refuses to make the room and requests another name. Otherwise, the Room Maker creates a room file in the user's Rooms Subtree.

### 5.7.1 The Room Maker Helper Processes

There is a collection of Helper processes, one for each room type, which are responsible for actually making rooms. Based on the room's type, the Room Maker creates the appropriate Helper process to make the room. The Helper inherits the Room Maker's window, and with it, asks the user questions that are needed to create the specified room type. After acquiring the necessary information from the user, the Helper fills the room file with Icon Descriptor information and sends an ADD NEW ICON request to the Room process to add a new door to the current room; this new door leads to the new room. If the room file cannot be filled with Icon Descriptor information or if the ADD NEW ICON request fails, the Helper removes the room file and displays an appropriate message.

## 5.8 The Exit Process

The Exit process is created by entering through the Lobby door icon. The Exit process makes an ENTER ROOM/FORGET ALL request with the location of the Lobby room file to the Room process.

## 5.9 The Entrance Activity

Entering through the Entrance door icon in the Lobby starts the Entrance activity. The Entrance activity creates a window and displays a list of user names. This list is acquired by looking directly underneath the root of the Users Tree. After the user selects his user name, he presents his password by typing. The Entrance activity encrypts the password and compares the encrypted password with the contents of the *password file,* located in the user's subtree (see Figure 5-3a). If the encrypted passwords do not match, the Entrance activity destroys itself. Otherwise, the activity makes an ENTER ROOM/FORGET ALL request with the location of the user's Office room file to the Room process.

## 5.10 References

[Cheriton et al. 1979]

D.R. Cheriton, M.A. Malcolm, L.S. Melen, and G.R. Sager, "Thoth, A Portable Real-Time Operating System", *Comm. ACM,* Vol. 22, No. 2 (Feb. 1979), pp. 105-115.

[Cheriton 1982]

D.R. Cheriton, *The Thoth System: Multi-Process Structuring and Portability,* Elsevier Science Publishing Co., 1982.

[Malcolm et al. 1983]

M.A. Malcolm, et al., *The Waterloo Port Programming System,* Computer Science Department, University of Waterloo, Jan. 1983.

# 6

# Summary and Further Research

This chapter summarizes the contributions of this report. Both the desktop and room models were discussed throughout Chapters 2, 3 and 4. Here, this material is consolidated and the fundamental differences between the two models are emphasized. For the design principles framework, several directions are suggested for pursuing and using the framework. With the help of both experience and our design principles framework, we prescribe some enhancements that can be made to *Room*. In closing, we briefly report the current status of *Room*.

## 6.1 The Desktop Versus the Room Model

An important design criterion of a user interface is that it should be *both* helpful toward the struggling novice and not encumber the expert. Reconciling these two often dichotomous goals can be difficult. The Xerox Star effectively demonstrates several principles that address the problems of reducing the overhead of learning. The main contributions of the room model are principles that improve an expert's productivity *without* compromising the principles embodied in Star. The following summarizes the techniques that the room model uses to promote efficient interaction.

### 6.1.1 Activity Switching

A user often switches his attention between activities. For example, during a session, a user may read and send mail, edit various documents and work on various programs. Because of its single workspace, the desktop model forces the user to rearrange or clear his desk to prepare for another activity. Depending on the number of tools and other objects associated with each activity, and the frequency of activity switching, a user may waste significant time rearranging his desk. In contrast, the room model permits the user to gather related tools and organize them into an efficient arrangement for accomplishing a particular task. Switching activities simply involves entering the room designed especially for the task. Hence, most of the overhead of switching activities is eliminated.

### 6.1.2 Organization

A consequence of not having to set aside objects with each task change is that objects tend to remain in the same place. This results in increased operating speed by reducing the time required to find an object. After having worked in a particular room several times, a user remembers the positions of icons and can quickly locate them without much conscious effort. In the desktop world, the positions of objects are more transient and the user must engage in time-consuming searches.

### 6.1.3 Tailoring

Universal commands have the effect of promoting consistency and hence, make a system easier to learn. However, generality incurs a cost: most tasks typically require *several* general steps. For example, to move an object a user must (1) find the object, (2) find its destination, and finally (3) invoke the move operation (steps (1) and (2) are themselves composites of general steps). Although the mechanics of this operation may be the best way to move an object in general, it is clearly not the most efficient if a particular object is frequently moved to the same place, such as to a printer; a single step is all that should be needed.

In Star, objects can be tailored but operations cannot. Hence, operations such as printing or mailing a document can only be accomplished by finding the document and moving it to the appropriate icon. In the room model, every icon, such as a printer and mail icon, can be either universal and able to operate on *any* document, or can be tailored to print or mail a *particular* document.

### 6.1.4 The Room Maker

Aside from the obvious problem of learning the capabilities of a new application, an expert can experience other problems when placed in a new task domain. The first problem he encounters is finding the various tools which have been developed to aid him in his new work. The Room Maker eliminates the tool gathering stage by constructing a room that contains all the available and relevant tools to perform a task, and arranges them in a standard fashion. Not only are the tools provided, but each tool is tailored specifically for the particular task.

## 6.2 Using the Design Principles Framework

In Chapter 4, we derived user interface design principles from the precepts of learning psychology. Our work is a first approximation and hence the framework must be refined and validated. It must also be extended with principles from other facets of psychology such as memory and perception.

The framework can be used as a basis for evaluating designs. For example, the Trial Space Reduction principle asserts that learning time is inversely related to the size of the trial space. With trial space size measurements of various user interface designs, a designer can apply heavier preference weights to those designs with smaller trial spaces.

There are often tradeoffs with applying a design principle. It is important that a designer properly evaluate these tradeoffs; otherwise, a principle may degrade rather than enhance performance. In this respect, the principles in our framework are qualitative in nature. A logical and necessary step for the framework is to develop quantitative models that reveal relationships between these principles.

The framework has been quite useful in the design of *Room* and other Port applications. At the very least, it can serve as a checklist that provides a systematic approach to building user interfaces. If only for this reason, we believe that the framework is worth pursuing.

## 6.3 *Room* Enhancements

### 6.3.1 Removing Rooms

*Room* currently lacks a facility for conveniently removing rooms. The problem does not lie in removing the room file (which is not difficult) but in detecting and hence preventing the removal of an intermediate room. An intermediate room is an essential junction for reaching other rooms. Removing such a room could leave a user stranded in a room with no route to his Office, or it could permanently sever all access to a vital collection of rooms.

Another problem with removing a room is the removal of all doors leading to that room. The ability to copy and distribute doors complicates the door removal process as all room files must be searched for "dead-end" doors. This would be possible if door distribution were limited to only the same workstation. However, *Room* permits doors to be distributed to other devices (e.g. the Truck) and even to other workstations. A possible solution would be to remove all dead-end doors from only the user's private collection of rooms on his workstation. The remaining dead-end doors could eventually be eliminated by removing a dead-end door when a user attempts to pass through it.

### 6.3.2 A Higher Level for Manipulating Rooms

Aside from the aforementioned difficulties of room removal, there is the more important consideration of designing the logistics of room removal at the user interface level. The *Room* environment lacks a means to visually identify and operate on a room as an individual object. Applying the Dimensional Expansion principle, a higher level application that can manipulate rooms is needed. The task of making and removing a room would then be relegated to this application.

This application could enable the user to view the "building" of rooms, showing their relative locations and connecting passageways. The removal of a room would entail selecting the room and invoking the REMOVE operation. Adding a room could entail selecting the room to which the extension is to be connected and invoking the BUILD ROOM operation. Other operations such as moving a room and copying the contents of a room could be included. This level could also be used as a quick way to travel between distant rooms, and perhaps eliminate the need for door copying.

The problems of displaying room layouts may be as difficult as those encountered in designing masks for integrated circuits. However, with suitable constraints (e.g. rooms must be square and have at most four doors), room layout could be made feasible.

### 6.3.3 Applying Cosmetics

The use of metaphors can reduce the time it takes to acquire concepts. The development of visual cues to accentuate these metaphors would also aid the cause. For example, learning the notion of a door could be expedited if the door icon were to visually "open" when entered. The notion of "traveling through" a door could be strengthened by "expanding" the door to reveal the contents of the next room instead of simply redrawing the display.

Examining an icon visually replaces the room display. This closely resembles *Room*'s entrance into another room. As a result, some users have thought icon examination meant "entering" the icon. This problem could be reduced by having the icon "grow", occupying as much as the room display as necessary; partially covered icons should be visible "behind" the "opened" icon.

The visual quality of the icon pictures in *Room* definitely needs to be improved. The available display device restricts the quality of the pictures. A display device with richer graphics capabilities would relieve this restriction and better equip *Room* to incorporate principles from perception psychology.

### 6.3.4 Stimulus Differentiation for Icons and Rooms

Many users of *Room* have been confused about the distinction between door icons and activity icons. Door icons should look different than activity icons because they do not behave in the same manner. As this violates the Consistency principle, doors should perhaps be made into objects which are different from icons.

The room is another possible candidate upon which to apply the stimulus differentiation principle. One type of room should look different from another type. For example, a Truck should appear different from a Supply room.

### 6.3.5 Eliminating the Entrance Activity

First time users have an especially difficult time assimilating the Entrance application into the room model. The visual cues of the Entrance application are poor and consequently interfere with the room metaphor. Because the Entrance application takes over the entire screen and visually replaces *Room*, many have thought the Entrance to be another room, albeit atypical.

One way to reduce this confusion would be to integrate user authentication into *Room* rather than delegate it to another activity. Each user of the workstation would have an Office door in the Lobby. Overpopulation in the Lobby can be alleviated by moving the Office doors of a user group to adjacent user group rooms. To enter his Office, the user would find and enter his Office door. *Room* would prompt the user for his password without disturbing the display in a drastic manner. This should appeal to the more familiar notion of a *locked door* and eliminate some of the confusion caused by the security requirements of a workstation.

### 6.3.6 Eliminating Attributes

It is not clear whether the concept of icon attributes is necessary. To dispense with this concept would reduce the learning overhead of *Room*. Of the three options – removable, copyable and movable – removable is the one most often used. The other two have been more useful to the workstation administrator in protecting icons in public rooms, such as the Supply Room and Lobby. The safeguard against accidental removal offered by the removable option could be provided with an extra set of operations; that is, selecting the REMOVE operation should result in another set of operations – REMOVE and DONT REMOVE – asking for confirmation.

The ability to edit information in the *Room* window would obviate the need for both the name and contents attributes. The help and activity attributes are more fundamental. It may be possible to absorb them into parameters or simply revoke the ability to alter them once created. The former approach could be applied to icons that the user creates. The latter could be applied to icons that the system provides since both the help and activity attributes do not change.

### 6.3.7 Accommodating Network Services

The Port operating system enables a service to be used across a network. Network services differ from local services in one very important respect – the failure of either the machine offering the service or the network connection renders a network service inaccessible. A useful extension to *Room* would be the provision of availability indicators on icons. This would enable the user to determine the availability of a service by simply inspecting the appearance of the icon. There would be no need to start an activity to acquire this information.

A user may have Offices on more than one workstation. *Room* should permit a user to enter rooms on other workstations in the network.

### 6.3.8 Integrating *Room* into the System

The Port operating system had its own self-sufficient command interpreter well before the inception of *Room*. Consequently, *Room* has had to conform to some decisions made without *Room* in mind. For example, Port commands are designed with few options to maintain simplicity and reduce typing. Complex operations are possible with the help of simple commands and the *pipe* facility. However, in the *Room* environment, selecting options is both mentally and physically simpler than constructing pipelines. This is because a complete list of options for an activity icon is displayed, whereas a list of commands and their options is not displayed. For *Room* to be more effective, it may be necessary to integrate certain popular pipelines into more powerful commands.

## 6.4 Current Status of *Room*

All features of the *Room* environment as described in Chapter 3 have been implemented and tested. *Room* is used in a fourth year undergraduate course. So far, approximately 100 students have been exposed to the system over the course of 8 months. Moreover, all of *Room's* features, save the Room Makers, are used commercially by a number of companies; of the Room Makers, only the empty Room Maker has been made available commercially.