THE UNIVERSITY OF CALGARY

USER MODELING IN INTERACTIVE COMPUTER SYSTEMS

by

SAUL GREENBERG

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

FEBRUARY, 1984

Abstract

This thesis discusses user modeling in adaptive interfaces in detail; presents a taxonomy of user modeling; identifies the dominant issues; describes an application amenable to adaptation; and investigates, through human factors experiments, the viability of adaptive systems.

The term "user modeling" is examined first. A framework for characterizing adaptive interfaces is proposed, first through the presentation of two possible system architectures, and then through a taxonomy which classifies modeling into three types. The literature on fundamental issues in adaptive interfaces is then surveyed, but little is found in the way of empirical studies. In particular, the most fundamental issue — whether or not adaptive user modeling is a viable alternative to non-adaptive systems — remained unanswered.

Repetitively accessed data bases, an application area amenable to personalization, are defined and examined through case studies. A personalized telephone directory system, built on an alphabetically-ordered menu interface, is constructed following guidelines derived from this work. The directory system is used as a test bed for examining the viability of adaptive interfaces. The results suggest that adaptive systems — at least in the personalized directory — can be far superior to non-adaptive interfaces, thus refuting many of the arguments found in the literature against personalization.

Acknowledgements

A thesis is rarely the work of one person, and this one is no exception. There are many friends and colleagues, too numerous to mention, who supported me with their ideas, time and patience. But I must give very special thanks to Ian Witten for being more than I could ask for in a supervisor, to David Hill for his contagious enthusiasm in the man/machine interface field, to Barb Gosling, Bruce Dunn, Geoff Powter and Don Chandler for opening the back door of experimental psychology, to Blob for forcing me to put aside my work for the sake of the mountains, and, of course, to Judy, who makes it all worthwhile.

Table of Contents

List of Tables

List of Figures

1. What is User Modeling?

The reasonable man adapts himself to the world: the unreasonable one persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable man.

(George Bernard Shaw 1856 — 1950)

People are different. Each one of us has idiosyncrasies, preferences, dislikes and a myriad of habits which characterize our personalities. We drive distinctive cars, wear conservative to outlandish clothes, have varied tastes in friends and believe in highly personal philosophies. Yet we deal readily with this variety. We know what behavior is generally acceptable and we interpret individuals' actions according to our own perceptions and past experience.

The complex dynamics of human interaction are taken for granted by most of us. We are continuously involved in communication, whether it is the subtle intermingling of voice, intonation and body actions of face-to-face conversation or the passive absorption of one-way broadcasts, such as television. Considering the diversity of our backgrounds and makeup, it is surprising that communication does succeed.

Simplistically, we comprehend one another because we make a model based on how we perceive the other person's behavior, and then try to interpret his actions according to that model. For example, we immediately greet and freely exchange information with a friend, for our model of him includes trust. Yet communication with a business competitor is guarded, for our model warns us that ulterior motives may exist. Indeed, entire classes of speaking styles exist, ranging from the formal language of strangers to the jargon of peer groups to the intimate interchange between lovers.

The computer, on the other hand, mimics human behavior. It carries on two-way conversations, often flavored with the "personality" of the system designer. As such, it elicits highly personal responses from its users, which are not reciprocated. The computer interprets human actions in a well-defined, highly rigid manner, and reacts uniformly to all users.

How do people converse with computers? Initially, the only mental model we have of the machine is that supplied through instructions, preconceptions and by our previous experience. As our familiarity with the computer grows, we build a working model of what personal behavior the system requires of us, and an expectation of how the machine should respond. If all goes well, the final model will closely match the system's capabilities. However, the computer is a general purpose machine, often running systems designed for diverse objectives with minimal commonality. The model we have for one computer task may be totally inappropriate for another, even though the physical interface, such as the terminal, is identical. Thus we learn to adapt to different computer systems as we learn to adapt to different personalities.

The modeling task, however, is rarely reciprocated. A machine has no capacity for understanding human variability. The computer usually employs a single algorithmic model which reacts to any user. The onus is then on the user to understand that inflexible and probably inadequate model. User frustration and/or failure at this chore may cause him to cease further interaction.

User modeling is defined as a set of rules, formulated by the system designer, which the system follows to determine its reaction to a user. In other words, it is the computer's model of the user. A simple program may respond to given input in a manner specified by the model by recognizing an input string as a user's command. Unfortunately, computer systems normally include a highly constrained, non-adaptable model of the user, which each user must fit if successful interaction is to occur.

This thesis speculates that the user model can be adaptable. Within limits, the computer can examine the communication channel — such as the user's input — and build or modify a model founded on rule-based inferences. It can then use this model to predict what the user is likely to do and alter the interaction with the aim of easing the user's labor. The computer becomes adaptive, in that its "behavior" towards the user changes as the user model is constructed. Ideally, the computer's behavior change not only eases the user task, but also remains consistent with the model the user has of the computer.

1.1 Aim and Scope of the Thesis

User modeling in interactive systems is ill-defined and seldom studied. This is not due to any serious conceptual barriers but rather to the infancy of the subject. The main objective of this thesis is to determine the viability of an adaptive interface in an interactive computer system, thus providing a concrete refutation of objections raised by certain authors (to be discussed in Chapter Three). More specifically, it has four objectives. First, it attempts to define user modeling in categorical terms independent of the underlying support theory. The system built upon finite automata principles, artificial intelligence techniques, or experimental psychology's recommendations should all find a niche in the taxonomy, for the ultimate objective in all cases is the improvement of the man/machine interface. The second objective is to discuss general issues in user modeling through a brief literature review. This not only highlights the controversy surrounding user modeling, but also illuminates the lack of empirical support for the various points of view. Thirdly, a specific application field amenable to adaptive techniques is considered and analyzed for tentative guidelines to facilitate personalization. The final objective is to examine the effectiveness of modeling through experimental evaluation of a system constructed according to the guidelines.

The remainder of this chapter introduces two adaptive system architectures, followed by the description of a modeling taxonomy. The architectures provide physical categorizations of the modeling interface, in which the user, computer and designer are seen as system components; whereas the taxonomy examines the ways these three components interact.

Chapter Two presents five case studies of selected systems that illustrate the concepts presented in the fundamental modeling taxonomy. Although the chapter focuses on specific applications, the related criticisms are usually fundamental to the modeling type chosen.

Chapter Three examines general issues associated with adaptive systems. Why model the user? Is there any gain from doing so? The opinions are as diverse as the authors, for there are few quantitative studies available to support or disprove the various viewpoints. The taxonomy of Chapter One and the case studies of Chapter Two supply a framework for critically examining each point presented. The most fundamental question left unanswered by the literature is whether or not adaptive systems can actually ease the user's task.

The remainder of the thesis attempts to answer this question, first by studying user habits in a specific application domain and then by proposing and testing an adaptive system based on these findings. The overall goal is to support the viability of the user modeling idea by constructing a successful system. Although this does not guarantee the success of all adaptive systems, it does allow a definite rebuttal of general criticisms concerning user modeling as a whole.

Chapter Four introduces the concept of ***repetitively accessed data bases***, an information system ripe for automatic modeling techniques. A specific data base of this type is analyzed for users' access characteristics, upon which specific modeling guidelines are based.

Finally, Chapter Five examines the effectiveness of automatic user modeling in facilitating transactions between users and computer-driven telephone auto-dialers. An adaptive system is proposed, based on the guidelines of Chapter Four. Human factor experiments then compare subjects' use of the adaptive and non-adaptive systems. The final results suggest specific application guidelines as well as supplying empirical evidence to support the user modeling concept.

1.2 Architecture of Modeling Systems

There is, as yet, no consensus of what actually makes up an adaptive system. Subtly different viewpoints may give rise to systems with basic conceptual differences. For example, a human factors specialist may design a system incorporating numerous psychological principles, an artificial intelligence disciple can consider it an expert system problem, whereas a theoretician may view the man/computer interaction as a state machine. Even peers within a discipline describe their ideas with dissimilar jargon and architecture. Two different models of adaptive systems are reviewed in this section (Edmonds, 1982; Innocent, 1982), making broad and specific comparisons between two authors' concepts.

The study of man/machine systems regards the user and the computer as two physical components of a total system. However, the computer operates in electrical engineering or technical computing terms, whereas a user's operations are incompletely described by psychological theory. An interface which acts as a "mediating component" is obviously necessary to allow the two to interact in a meaningful way. Simply stated, it must discover what task the user wishes to accomplish and pass this information on to the computer for execution. Conversely, data received from the computer must be transformed into a form understandable and relevant to the user. However, the mediating component is much more than a simple input/output channel. Considerable processing may take place in the conversion of input/output — such as speech or natural language — to or from an intermediate machine form, followed by its analysis for secondary information.

Two authors, Edmonds (1982) and Innocent (1982), support these principles. Edmonds (1982) describes the mediating component as an *interface processor* between the user and the *background processor* (i.e. the functional machine), whereas Innocent (1982) calls it a *facade* which isolates the user from the system (Figure 1.1). It appears from the Figure that each is using different jargon to describe an identical architecture. However, further breakdown of the mediating component reveals fundamental differences in the construction of the user model. Edmonds (1982) considers the mediating component to be general-purpose, in which the designer, user or the system can modify the user model. Innocent (1982) describes it as self-adaptive, in which the system automatically changes the interface according to its perception of the user state. The conceptual differences between these two architectures are described below.

Edmonds (1982) views the system components as *processors*, implying that all actions may be performed in parallel. The mediating component is called an *interface processor*, while the remaining system is termed a *background processor* (Figure 1.1). The interface processor is made up of *i/o processors* performing mechanical transformations of the input and output, and a *dynamics processor* which determines the action to be performed based on internal model parameters (Figure 1.2). Adaptation of this internal model is permitted through the use of a *monitor processor*. Its usual role is to record information found in the interface processor. The designer/evaluator may modify the interface himself using this information or the monitor may alter the dynamics processor's parameters. The rich interconnection between the task hierarchy and the isolation of the processors such as input/output, performed action, and an "external" monitor is shown in Figure 1.2. Of primary interest here is the isolation of the background processor from any other system component — including the designer — through the interface processor. The centralization of the interface processor makes it the keystone of the system architecture.

Innocent's viewpoint (1982) envisions the mediating component as a *facade*, defined as the system as it appears to the user (Figure 1.1). A "hard" facade cannot be altered, whereas a "soft" facade can be easily tailored by the designer, user or system. Whereas Edmonds' design deals with all types of system adaptation, Innocent (1982) proposes a network specific to self-adapting interfaces (Figure 1.3). Innocent's self-adaptive user interface contains several components within the facade architecture. An *i/o module* within the soft facade processes input and output into an appropriate intermediate form. The *monitor module* sifts, processes and transports any relevant information to the *expert modifier*, which recognizes the significant information and recommends changes to optimize the facade. The *control module* accepts these

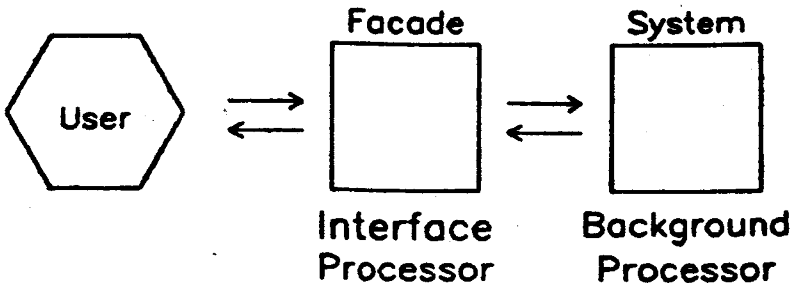

Figure 1.1: An architecture overview

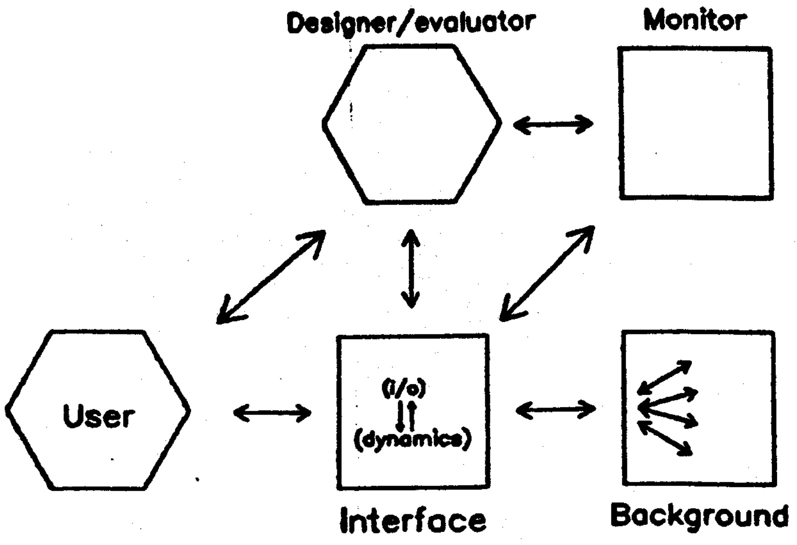

Figure 1.2: Architecture detail (Edmonds, 1982)

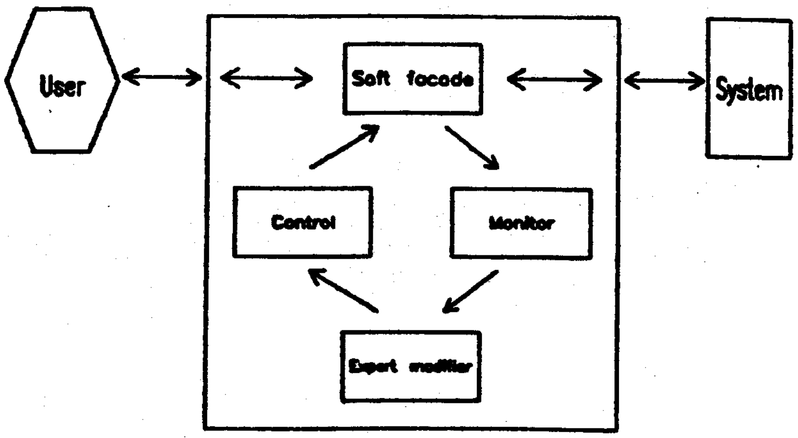

Figure 1.3: The self-adaptive user interface (Innocent, 1982)

recommendations and updates the actual facade.

The key difference between the two architectures is that the monitor is an integrated part of Innocent's (1982) self-adaptive user interface. This implies that all changes to the user model are executed without any direct user/designer control. The system is in complete charge of updating the model.

In summary, the examination of two architectures of adaptive interfaces shows that although differences do exist, similarities in overall components are found, particularly in the isolation of task-oriented interface units.

1.3 A Taxonomy of User Modeling

A system which adapts is one that changes to fit the current model of the user. But which system component is the one responsible for actually modeling the user? The dynamics of man/machine interaction is subtle and ill-defined on the user side but extremely precise for the machine. A person rarely looks at a system in isolation. Rather, he views it through a collection of related experiences in an attempt to understand it. The machine, on the other hand, both presents itself and interprets outside actions through the stringent rules laid down by the designer. The interplay of the user, machine and designer suggests different methods of implementing user modeling.

This section will present a taxonomy of adaptive systems, in which the emphasis of the modeling task is placed upon either the designer, the user(s), or the machine. Each has advantages and constraints. The designer's model creates a general machine for the general user with little malleability for adapting to individual needs. A user's model is highly personal and effective, but suffers from requiring explicit construction. A machine's model is also personal, but it must construct the model by the limited rules laid down by a designer. Although the taxonomy will describe each in detail, it must be understood that the boundaries are hazy. A well-designed system will likely use combinations of different modeling techniques to present the best adaptive system.

1.3.1 Canonical modeling — Designer models the user

The term ***canonical*** is defined as "conforming to the general rule" (Merriam & Webster, 1964). A ***canonical model*** is the model of the single, "typical" user which attempts to represent the general population[1]. The design and application of canonical models is explored in this section.

The first principle, and perhaps the golden rule of man-machine interface design, is to "know the user", first stated by Hansen (1971) and echoed by myriads of other authors (Pew & Rollins, 1975; Maguire, 1982; Gaines & Shaw, 1983). Authors of human factors guidelines usually interpret this as indicating a need for the designer to form a profile of the user for whom the system is intended to serve (Maguire, 1982), which is continually updated through a three-part dialogue between the dialogue engineer, the user and the computer as an intermediary (Gaines & Shaw, 1983). Although these authors accept that the ideal user profile is generated for the individual, most designers settle for fashioning a single canonical framework representing all users.

[1] The term ***canonical model*** was coined by Rich (1983).

The first and still the most prevalent method of constructing a canonical model is through intuition. In early systems, the maxim of design was: "what kind of interface would I like to see?". The result was often a dialogue the designer loved and others just tolerated. A typical example of problems that occur through this approach is in vocabulary usage. For example, an empirical study of natural command names in text-editors found that novice users do not use the same language as system designers to describe text editing tasks (Landauer *et al*, 1983). Furthermore, the investigation concluded that intuitive guesses by designers for "common" and "natural" command names are likely to be hazardous. Further difficulties arise in other domains. For example, the designer may personify the system with aspects of his own personality — such as insults in error messages — which are not appreciated by tired and overworked operators. That systems do succeed is not really a measure of system designers' insight, rather it is an indication that users are able to adapt even to very ill-conceived dialogues.

A second method of canonical model building is through analysis of the users: their needs, usage patterns, vocabulary, and surrounding environment. Shneiderman (1979) succinctly states that:

> System developers are increasingly aware that *ad hoc* design processes, based on intuition and limited experience, may have been adequate for early programming languages and applications but are insufficient for interactive systems which will be used by millions of diverse people.

The conscientious designer will usually follow the myriad of available dialogue guidelines to build the typical population profile. Constant feedback from users will occur during the design and installation phase. He will realize that user needs do not remain static and will thus follow Gaines' & Shaw's (1983) recommendation that "design never ceases". A three way dialogue will constantly take place, possibly through use of a mail system, through logging of activities or through direct conversation with the population or its representative. This view is illustrated graphically in the previous section by the inclusion of a *monitor* in the Edmond's (1982) interface architecture (Figure 1.2).

The designer must be cautious in following a prescribed set of guidelines — especially if they attempt to be universal — for they may not be suited to the intended system users or application. Current literature is now attempting to suggest characteristics of certain target populations and system types. For example, computer experience is considered a measure in defining user group names such as naive, casual, novice, regular, secondary, expert and system designers. Although the actual boundaries between these groups are ill-defined, they do provide a rough scheme for user taxonomy. Cuff (1980) explores characteristics of the *casual* user and provides loose recommendations on what features the dialogue should provide. Kennedy (1975) looks at some behavioral factors affecting the training of *naive* users. User and task specification combined with human factors experimentation may eventually furnish a framework for combining good canonical models with good system design, although certain issues (to be described in Section 3.1.1) must be addressed.

In the final method of canonical model building, the system implementor relinquishes control over the design process by allowing a higher degree of user involvement. Eason & Damodaran (1979) distinguish five different levels of user involvement in the design process, ranging from system-centered to user-centered design (Table 1.1). Of course, problems do exist with user involvement. Both designers and users require the ability, tools, sympathy and

System-centered design	— None
	— Communication, consultation & training
	— User representatives
	— Participative design
User-centered design	— Users design, experts advise

Table 1.1 Types of user involvement

motivation to interact. User demands may also lead to so many compromises in attempts to appease individual desires that the final model will consist of a smorgasbord of personalities rather than a proper canonical representation, thus jeopardising its effectiveness. However, user involvement is usually viewed as an asset to the system design, providing that the interaction process is effectively controlled.

Canonical modeling will have great importance in the design process for years to come. The techniques of building these models are currently being refined through user involvement and the experimental process. Formal methods of system/design which include the user as part of the system specification may eventually replace the *ad hoc* processes of the past.

1.3.2 Explicit modeling — User models himself

Liberty means responsibility. That is why most men dread it.
(George Bernard Shaw, 1856 — 1950)

Canonical modeling was seen to encompass the frequently used strategy of having a system designer interact with the user population and adjust the system to fit the current need. Unfortunately, its view of the user as "typical" cannot be accurate for a highly heterogeneous community (Rich, 1983). Edmonds (1982) suggests that our knowledge of human behavior is inadequate to portray correctly a canonical user, especially one whose needs will change over time. An alternate "personalized" outlook shifts the design focus away from the canonical view towards a collection of models of individual users. Explicit modeling is one possibility; automatic modeling — considered in the next section — is another.

Explicit modeling is defined as the ability of a user to provide explicitly a limited model of characteristics and/or actions the system is to follow in a defined situation. It is not feasible for the designer to mould a complete model of each user, especially if it is to evolve over time. However, the designer can distinguish, through consultation with the user group, the areas of the system model that should retain canonical attributes from those controlled by the individual's model.

It must be stressed that a "model" of a user is not a simulation of a person. Rather, it is merely a set of very limited guidelines which the system will follow. The model complexity can range from simple definitions found in an abbreviation table to the more sophisticated interactive eliciting of personal constructs (Shaw, 1980) which may formalize the modeling process (Gaines and Shaw, 1983). Architecturally, the interface detail described in Section 1.2 (Figure 1.2) would have a strong, two-way link between the monitor and the user, lessening the role of the system designer as an intermediary.

Any explicit adaptation of a system by the user provides an example of the user modeling himself. Specifically, the advent of interactive systems forced users to adapt to the designer's concept of a good interface. The mechanical and often confining nature of this interface caused programmers to write utilities to perform tasks specific to their own needs. In time, the newer generation of "user-friendly" operating systems provided personalization facilities: for instance, user-defined abbreviations, start-up profiles executed upon logging in, and the ability to change system parameters — such as the prompt — at any time. Unfortunately the progression has, in some cases, introduced a new complexity of problems. The sub-systems created for facilitating the user's construction of a model may be complex systems in their own right, possibly negating any benefits.

A typical explicit modeling sequence usually consists of three phases; an idle phase, a setup phase, and a quasi-stable phase. Initially, a new user goes through an idle phase when learning the task-specific portion of the system. During this period, the system follows a default canonical model provided by the designer. After gaining some confidence in the application, the user will enter the setup phase, in which he explicitly fashions a model to replace the default. Finally, a quasi-stable period is reached, in which minor adjustments to the model detail are performed, usually reflecting task changes and the user's increased sophistication. Application of the model is performed during all three processes, either explicitly by the user or automatically by the system.

Each phase is now examined for its own particular attributes. The idle phase is a particularly sensitive period, for the user is at the mercy of the default model. Ideally, a sensible canonical model — as described in the previous section — is provided by the designer. Unfortunately, this is rarely the case. The user must therefore struggle through the system without the benefits of an explicit model at a time when he most needs it[1].

The setup phase usually begins after a user has gained a certain level of proficiency in the application task. Usually, he discovers through documentation or other users that the default model can be altered. The user will then attempt to fabricate a profile by taking time out from the application task to learn the syntax and semantics of the explicit modeling protocol. Conversely, he may just dabble at it through trial and error methods. A severe disadvantage of explicit modeling is illuminated; the user has an overhead of extra time taken in the setup process which detracts from the actual application. Rich (1983) warns that "people do not want to stop and answer questions before they can get on with whatever they are trying to use the system to do". Because of this, some users never even enter the setup period. In a well thought

[1]The Unix system has a "$" as a default prompt which can be changed at any time to a more meaningful phrase, such as the machine name i.e. "Vax" or the system name i.e. "Unix" But what does the default "$" signify? Novice users are often disoriented in their location when switching between the editor and operating system. A new user may even think he has to enter the amount of money he has available for accounting purposes!

out system, this time may be minimized; perhaps by eliciting the model through interactive means and thus eliminating the need to learn syntax. Another problem with certain systems is that the default model is disregarded once any explicit model entry has been made, thus making it impossible for the user to create the explicit model piecemeal[1].

The final quasi-stable phase usually lasts for the lifetime of system use. It is a "stable" period in the sense that the primary model is already in existence; it is "quasi" because the model is usually in a state of flux. When user needs and sophistication change, so must the details of his model. As with the setup period, time must be taken out from the application task to execute these changes, an overhead that may be too great for short term needs.

Consultation of a defined model may be explicit or implicit. An example of explicit use is an abbreviation facility, where the user must remember the abbreviation and its meaning in order to use it. An example of implicitly consulting a model is a profile that is executed upon an action recognizable to both the default and the modeled environment. Unfortunately, this can surprise the user who has forgotten the altered way the system will react to a given action.

An attempt to outline the explicit modeling process was made by defining three phases. However, no sharp boundary exists between these phases; and the flow through them is not necessarily sequential, for the user may discard or totally overhaul the model at any time. The presence of any of the three phases depends on the particular system design, which may eliminate any of them or construct the system in a transparent manner. Specific illustrations are presented in the next chapter to provide examples of certain modeling techniques and the advantages and disadvantages that characterize them.

1.3.3 Automatic Modeling — System models user

"It's a poor sort of memory that only works backwards," the Queen remarked ...
(Through the Looking Glass — Lewis Carroll)

In explicit modeling, a user requires time and effort, extra to the primary task, in order to create or change the model the system is to follow. Automatic user modeling eliminates this secondary task by having the system automatically form the user model.

Automatic modeling is defined in two stages. First, it is the ability of a system to build a limited model of a user's characteristics continuously. Secondly, this profile is consulted by the computer at the appropriate time. Although there is a rough analogy in these stages to explicit modeling phases, significant differences do exist, especially in the dependence of the model on high probability guesses it makes of user habits.

[1]One editor called *emacs* (Stallman, 1980) has a highly sophisticated startup profile which controls virtually every component of the editing task. However, once the user creates his own startup, the default one is ignored. The hurdle of adapting to the following period of system unpredictability causes many users to avoid the setup process.

Model construction is based on the system monitoring some aspect of the man/machine dialogue. A hypothetical system configuration was shown in Innocent's (1982) self-adaptive user interface (Figure 1.3). All user/machine interactions are directed through this interface, which monitors events and updates the user model based upon its "knowledge" of human behavior (Innocent, 1982). The model may be fashioned from scratch, using the inferences as building blocks. Alternatively, the system inferences may categorize a user into stereotypes, facilitating model choice from a supplied range of defaults.

Stereotypes allow the system to predict many user characteristics based on minimal information. For example, a system may have dialogue models available on a continuum from novice to expert users. The main problem lies in discovering the user's current experience level. Primitive forms of deducing user classification may use total login hours and time between sessions as a rudimentary measure. Alternatively, terminal location can furnish the clues: an advanced dialogue has a higher probability of occurring in a system designer's office than on a public terminal. A more precise approach would monitor the dialogue and rank the user based on the sophistication of his/her queries. However, some mechanism must exist to resolve placement conflicts due to competing inferences, and to correct model inaccuracies that exist in the stereotypes supplied (Rich, 1983). A pilot system called Grundy, built by Rich (1983), has the ability to stereotype its users according to book preference. Grundy's task is to recommend novels to people, based on its internal data describing books, and its collection of stereotypes containing facets relating to people's taste in books. Grundy's recommendations were found to be significantly better ($p<10^{-9}$) when using stereotypes than without them.

The second modeling approach constructs, rather than selects, a user model based on his system interaction. One example is a simple model of a menu-based system ranking menu items by each user's selection frequency. The frequently chosen items are then preferentially displayed before all others. A second example is Teiresias, a complex expert system which facilitates the changing of its knowledge base interactively when an error or inconsistency in the rule-based model is discovered (Davis, 1979). The primary difference between these two examples is that the first is constantly and continuously trying to maintain an accurate view of heterogeneous and dynamic users, whereas the goal of the second is to build a quasi-static model of an expert in a given field, a model destined to be non-adaptive for the end user.

The important implication derived from either method of automatic model building is that the information contained in the model will be guesses (Rich, 1983). Because of this, some mechanism must exist which ranks all components in order to update the model as new information comes in. For example, an initial period of high instability may exist upon first contact between the user and the self-adaptive interface. At first, a system adapting to user interaction will base much of its model on minimal input. As new information arrives, the "uncertain" model will change radically. Due to this, the user may envision the virgin system as unstable at a time when stability is crucial to the acceptance of the system. A *monitor* and *expert modifier* must exist to arbitrate the probability of model inaccuracy with model application. Yet another problem is the incorporation of a user's errors into the model. Unless the monitor can identify a segment of the dialogue as being a user error, the system will think that it is part of the user's normal procedure. Checks and balances must therefore be performed continuously, in which low-probability "facts" are discarded.

The second stage in automatic modeling is model consultation at appropriate times. This may be totally or partially automatic, either case resulting in an action visible to the user. For example, cursor location next to the most frequently chosen menu items is a totally automatic consultation of the model. A partially automatic system example queries the user for acceptance of auto-completed strings modeled on previous input. Either way, serious problems can occur when an action is performed at an inappropriate time, for negative effects on the user's view of the system may arise.

In summary, model construction and consultation were described as two usually distinct phases occurring in adaptive systems. Inferences of user activity can trigger the selection and update of model stereotypes. Alternatively, the inferences are used to continuously modify the system's current knowledge base of the user's characteristics. Rigorous checks and balances in model modification and consultation are necessary for a system to appear consistent and sensible to the user.

A variation of automatic modeling is combination modeling, a composite of explicit and automatic modeling techniques. It attempts to bridge the hazy categorization differences that will likely exist in actual implementations. Modeling generally consists of two distinct phases; model formation and model use, with explicit systems placing the onus of activating both phases on the user while automatic systems usually execute these stages implicitly. Combination modeling removes these limitations by allowing a fusion of both techniques.

Combination modeling, in its most realistic form, fashions the environment automatically and adaptively, yet remains passive until the user explicitly requests its use. This "combination" minimizes the user's setup overhead and inappropriate model invocation by the system, two severe disadvantages present in explicit and automatic systems respectively. The well-designed system will likely use combination modeling to minimize problems in adaptive systems. Issues such as user control, model accuracy, and the user's perception of the system will determine the proportion of explicit/automatic techniques in the composite system.

2. Examples of User Modeling Systems

A taxonomy of user modeling methodology was introduced in the previous chapter. Examples of selected systems embodying these concepts are now presented to illustrate some of the techniques used to implement these user modeling ideas. Where possible, the system is analyzed informally for the user benefits and drawbacks that accompany the modeling technique.

Six examples in all are offered; two on both explicit and combination modeling, and two on automatic modeling, where the second is proposed and discussed in detail in Chapter Five. The reader should already be familiar with canonical systems, as the overwhelming majority of present-day computer packages are canonical by default — no examples of this type are given. In contrast, automatic systems are a rarity; it is difficult to find enough examples to allow comparison of salient features.

These are selected examples. The emphasis is towards non-intricate systems; we attempt to avoid those with features that obscure the modeling focus. Where possible, a simple example of a particular modeling type will precede a more complex one.

2.1 Explicit Modeling — the Unix *Alias* System

Abbreviations are available on most operating systems in either user-defined or designer-defined forms. Practically all current command driven operating systems have designer-defined abbreviations for command names, although some allow only the short form to be used. Human factor guidelines for abbreviation methodology have already been proposed in the literature by Ehrenreich (1981), who mentions important issues such as truncation versus contraction, mnemonic meaningfulness, and length of abbreviation versus length of command. The availability of user-defined abbreviations is a common explicit modeling technique found on many of the so-called "user-friendly" operating systems (such as Unix and Multics). The abbreviation facility in Unix is briefly examined as an example of an expert-oriented system containing explicit abbreviation techniques, with user comments illustrating some advantages and disadvantages of self-defined abbreviations[1].

The Unix command-line interpreter has two facilities for allowing command abbreviations to take place; the execution of a ***shell*** file which contains an arbitrary set of commands, and the use of ***alias***, an explicit mechanism for creating abbreviations. ***Alias*** will be the method discussed, as it is more specific to the abbreviation task than shell files. The ***alias*** mechanism is used:

[1] The Unix system described in this thesis uses the *csh* (also called C-shell) command-line interpreter.

> ... to provide short names for commands, to provide default arguments, and to define new short commands in terms of other commands. It is also possible to define aliases which contain multiple commands or pipelines, showing where the arguments to the original command are to be substituted.
>
> (Joy, 1980)

The applications of ***alias*** are diverse. It can be used to create synonyms to commands for users preferring their own mnemonics. For example, negative transfer across operating systems can be partially avoided by aliasing a single command name to identical actions. ***Alias*** also provides default arguments on frequently used commands. An illustrative task is a user who regularly sends mail to the same group of people. He may create an abbreviation called ***mymail*** which invokes ***mail Bob Bill Janet Harry George***. A third use of ***alias*** is as a macro combining a group of commands into one package. For example, an ***alias*** entitled ***whatsup*** may be a macro of four commands which when invoked tells the user the current date, who is on the system, his mail status and lists his daily memos. Another use of ***alias*** is to request at any time an alphabetically sorted list of all the abbreviations and their corresponding long forms.

Many positive and negative criticisms have been raised by Unix users in the Department of Computer Science at the University of Calgary. Some users use it rarely or not at all, on the grounds that too many abbreviations are not only confusing but difficult to transfer across systems. In addition, many Unix commands are terse and rarely need shortening. Another problem is the unavailability of abbreviated commands or the surprise effect of unknowingly typing an alias when working in another user's environment. ***Alias*** usually works only at the beginning of a line, with in-line expansion done only in exceptional cases. It is also insensitive to context, in that the user must take care to invoke an alias only at appropriate times. Some users report that they cannot be bothered with the overhead of setting up abbreviation tables or with learning the more exotic features of ***alias***. Despite the criticisms, accolades are generally given for the ease of use for the tasks mentioned above, particularly in the ability to supply a shorthand notation for long command lines.

Alias was found to have both benefits and problems. It illustrates how even a very simple explicit modeling technique can lead to a diversity of user opinion.

2.2 Explicit Modeling — the Unix *Emacs* Environment

An example of an sophisticated explicit modeling technique is a tool allowing the creation or tailoring of the working environment. In its most primitive instance an expert user may write a program to execute a task which is not normally part of the system library. Many systems do not even allow this limited tailoring to take place easily. For example, the TOPS-10 machine has different commands for executing system routines and user-defined routines. In its more advanced form, some "user friendly" operating systems, such as Unix and Multics, offer a high degree of flexibility in user definitions of his working space. Environment tailoring may be done through command scripts invoked by shell files, resulting in mini systems which need not resemble the operating system at all. Proponents of explicit modeling may indicate that full environment tailoring and the unlimited power it supplies to the user is the best method of personalization. An editor named ***emacs*** is examined to illustrate some of the issues that can arise when this occurs.

Emacs is an extensible, customizable self-documenting display editor developed at the Massachusetts Institute of Technology. The claims of the original *emacs* designers sound promising:

> Users are not limited by the decisions made by the *emacs* implementors. What we decide is not worthwhile to add, the user can provide for himself. He can just as easily provide his own alternative to a feature if he does not like the way it works in a standard system.
>
> (Stallman, 1981)

Unix *emacs* is a powerful and highly sophisticated window editor built for the Unix operating system by Gosling (1981) on the foundations of other *emacs* editors (Stallman, 1981). Although a complete description of *emacs* is far beyond the scope of this thesis, some detail will be provided to describe the explicit modeling tools available within this editor. A brief survey of actual usage patterns and its associated problems will follow the system portrayal.

Emacs contains two levels of operation; an editing level which provides the tools to do all editing tasks, and an environment level which allows the user to modify the *emacs* working environment to his particular needs. The editing level supplies the user with a wide range of commands and packages. Many specialized commands are available, the most popular being bound to keys prefixed by control or escape characters. Complete cursor control, multiple windows, buffers and file management, spell packages, and compilation features which allow viewing of error messages and their source location are just a few of the features present in *emacs*. Unfortunately, there is a price to be paid for this sophistication — *emacs* is quite difficult to learn in its entirety and is usually used only by Unix experts. However, the learning of *emacs* subsets has been found comparable in difficulty to other editors (Roberts and Moran, 1982). Coffee room discussions are frequent in the Department of Computer Science between people expounding the virtues of *emacs* and others criticizing it due to the hardship of acquiring proficiency in the large *emacs* command set.

The environment level of *emacs* provides facilities for the user to tailor the editing environment explicitly to his general and specific needs. These facilities are abbreviation modes, the setting of key-bindings to specific commands, and command extensibility via both macros and a built-in Lisp-like programming language called Mock Lisp. Abbreviations in *emacs* are more versatile than the previously described *alias* facility. They are expanded in-line after the person has typed the contraction, supplying the user with immediate feedback of the full expansion of the entered term. The user can also define abbreviations specific to any context he desires by creating abbreviation tables global or local to specific buffers, file types and directories.

Alteration of key-bindings is a simple way of dictating which commands are to be executed when a particular button or sequence of buttons on a keyboard is struck. It can be used to avoid negative transfer between different editors by retaining similar bindings, to set up function keys for specific tasks, to assign common meanings to cursor keys across different terminals, and to bind commands to keys according to the personal quirks of the user.

Macros repeat a series of actions. It is possible for *emacs* to remember a set of sequential commands and keystrokes as a macro and then execute that set as if the user had typed it in again. Macros are usually useful for highly repetitive localized tasks.

The most complex area of *emacs* environment tailoring is the programming feature. The original implementation of *emacs* uses Lisp as the editor programming language, because an extensible interactive system requires an interpreter and the ability for programs to access the interpreter's data structures (Stallman, 1981). Gosling (1981) created Mock Lisp — a pseudo Lisp subset — to satisfy the above programming language criteria. Mock Lisp allows a user to write almost any desired function specific to an *emacs* editing task. For example, there are functions which automatically format text and indicate errors when the user is writing a program in a programming language such as C, Pascal or Lisp. Other functions specialize in Nroff document preparation and editing directories. The scope is almost unlimited. With Mock Lisp, the user can create functions and bind them to keys in a manner which could radically differ from the default version of *emacs*. Unfortunately, it also burdens the user with the learning of an additional programming language.

Emacs is perhaps the most extensive explicit modeling system the author has encountered. Yet, its very power is its weakness. As previously mentioned, many people find even the editing level of *emacs* too difficult to learn. A quick survey of selected *emacs* users in the Department of Computer Science has found that only a few users, usually Unix experts, have been able to or have bothered to learn the features that allow for explicit tailoring of their environment.

Seventeen frequent users of *emacs* (where *emacs* was the primary editor used at least 50% of the time) were surveyed as to their opinion on *emacs*. Of these, five gave themselves ratings as expert users, one as a novice and the remaining eleven as just adequate in knowledge. All but one have a "profile" which overrides the default *emacs* environment. However, of the sixteen who had a tailored profile, only three wrote it themselves while the rest had copied it, sometimes third or fourth hand, from another source. Of these thirteen who had copied profiles, nine had modified some aspects of it. Twelve of the seventeen users reported changing the key bindings to different commands. The surveyed users were then asked about their knowledge of Mock Lisp. Eleven of the seventeen indicated that they had little or no Mock Lisp expertise.

The survey above is not statistically accurate as the sampling technique was biased towards heavy users of *emacs*. However, certain trends and comments were noted which indicate serious problems with the powerful explicit modeling tools available in *emacs*:

- Learning the explicit modeling techniques such as Mock Lisp was an overhead to the editing task most users could not or would not undertake.
- Use of Mock Lisp was generally restricted to copying and modifying an expert's Mock Lisp profile. Thus the most powerful of *emacs* features was almost turned into a designer modeling technique — the real experts in *emacs* were the ones deciding upon the best environment.
- Almost all users report some degree of difficulty when using a non-familiar version of *emacs*. This happens regularly when people are working together or when one is giving over the shoulder advice to another about the text being edited (such as a program or a document). The main problems indicated were the unavailability of certain commands and surprises that occurred when key-bindings were non-standard. Many users suggested a need for standardization of key-bindings or a way of restoring the *emacs* default state.

Use of the complete *emacs* command set was restricted to experts only. Very few others were able to use effectively the sophisticated environment tailoring tools that are available. Thus the power given to users to model *emacs* explicitly after their own needs was neutralized in most cases — users had either to use the default environment or to ask the system experts. This directly contradicts the previously-mentioned claims made by the designers. In addition, confusion arose when using different versions of the *emacs* environment. *Emacs* clearly demonstrates the problems and false expectations that arise when too much power is given to the user.

2.3 Automatic Modeling — the *Predict* Terminal Interface

Few methods currently exist for modeling the user automatically. One technique makes use of repetitive patterns or inherent redundancy in user input to forecast what he will do next.

Predict (Witten, 1982) attempts to determine via frequency counts what tokens (characters, words or strings) the user is about to type. Its predictions are based on the assumption that a great deal of redundancy is found in most tasks, such as repetition of command strings in the command level of an operating system, limited vocabulary of keywords and variables in a programming environment and the natural repetition of characters and text occurring in documents. The interface to *predict* is fairly straightforward — predictions appear in reverse video in front of the current cursor location. The user may then decide to accept part or all of a prediction or he may ignore it altogether by continuing his typing.

Predict is supposed to replace the more common abbreviation facilities, which suffer the disadvantage of the user having to know in advance what terms he will be using frequently (Witten, 1982). *Predict* also provides automatic completion of what would otherwise be truncated abbreviations of command names, supplying immediate feedback to the user. Finally, the tedium of typing oft-repeated large strings is reduced.

It is easy to make criticisms of *predict*. Predictions appearing and disappearing on the screen can be quite annoying. Touch typists rarely look at the VDU, and will not notice predictions. Speed typists seem to be slowed down by the cognitive decision necessary to accept or reject predictions. However, *predict* may still be used to an advantage by certain groups. For example, a handicapped person whose cognitive skills far surpass his physical typing speed will probably use *predict* productively.

The man/machine interface to *predict* is not necessarily fixed to the mechanism described above. Witten, Cleary and Darragh (1983) have described the *reactive keyboard*, a text entry device based upon the adaptive techniques found in *predict*. Whereas *predict* only displays the most likely prediction, the *reactive keyboard* would display a weight-ordered menu of prediction strings in a separate window. The user would then use a two-dimensional pointing device, such as a mouse, to indicate the partial or complete string desired for text entry. If the desired string or portion thereof did not appear on the first menu, the user would have the option of asking for another menu of predictions. Simulation studies have shown that the probability of the correct prediction occurring on the first menu lie between 0.69 and 0.998, depending on how the system was primed.

An unpublished human factors pilot study of ***predict*** was made by the author and others. Results indicated that ***predict*** did speed up certain typing tasks, such as entering a Cobol program, and slowed down others, such as entering English text. However, most subjects were under the illusion that their productivity had actually increased and generally expressed satisfaction with ***predict***. Thus there is a fine line between productivity measures and user satisfaction; some users may be willing to reduce overall typing speed if the task tedium is reduced. It becomes difficult to forecast the viability of ***predict*** in the real world. More extensive human factor tests must be done. Alternatively, the system could be placed in the public domain to undergo the primitive but perhaps more realistic selection process of the marketplace.

2.4 Combination Modeling — the Unix History System

Combination modeling can minimize certain inherent problems found in systems that are solely adaptive or explicit. Both ***emacs*** and ***alias*** have an overhead of explicit system state setup which discourage full system utilization by some users. ***Predict*** users suffer the lack of control over unwanted predictions flashing on the screen. A combination system could eliminate both these problems by fashioning the environment automatically and adaptively, and yet remaining passive until the user explicitly requested its use. A simple example of a combination modeling technique is the Unix ***history*** mechanism.

> ***History*** is run as a transparent front end to the Unix command interpreter.
>
> The ***history*** mechanism ... allows previous (Unix) commands to be repeated, possibly after modification to correct typing mistakes or to change the meaning of the command. The shell has a ***history list*** where these commands are kept, and a ***history*** variable which controls how large this list is.
>
> (Joy, 1980)

History automatically maintains a ***history list*** of a predefined length on all user-invoked commands in the current login session. Partial or complete command strings can then be re-used by the user when forming a new command.

An example will illustrate how ***history*** works. Let us say a user has typed in six commands since logging in. The automatically formed history list may look like the one found in Table 2.1. The number prefixing each command is its position in the history list. The user can now invoke ***history*** in a variety of ways:

- Typing **history** displays the list in the Figure.
- Typing **!!** repeats the last command (***mail smith***).
- Typing **!w** repeats the last command starting with a 'w' (***who***).
- Typing **!2** repeats the second command in the history list (***alias e emacs***).
- Typing **!$** repeats the last argument in the previous command (***write !$*** is the same as ***write smith***).
- Typing **!5:2** repeats the second argument of the fifth entry in the history list (***print !5:2*** is the same as ***print file2***).
- Typing **^smith^smythe** repeats the last command replacing smythe for smith (***mail smythe***).

Position	Command	Meaning
1	who	(who is on the system)
2	alias e emacs	(abbreviate 'e' for 'emacs')
3	roff file1	(display file1 as a document)
4	rm file1	(destroy file1)
5	e file1 file2	(edit file1 and file2)
6	mail smith	(send mail to smith)

Table 2.1 History list containing six commands

Most of the basic features of *history* are illustrated in the above list.

Like other systems, *history* has its problems. Although the user does not have to set up the history list explicitly, he must still learn the arcane syntax necessary for its invocation, a syntax which is not analogous to anything else in Unix. Thus, the notation rapidly becomes quite cryptic, especially when using a mixture of arguments and commands. For this reason, many users commonly restrict themselves to simple uses of *history,* such as repeating one of their last few commands or correcting spelling errors. Another *history* drawback is its reliance on the undependable nature of human memory. The sequence supplied in Table 2.1 will illustrate an example. The user has displayed a file, destroyed it and edited a new one of the same name (positions 3 — 5). After completing the editing task and sending some mail (positions 5 and 6), he may wish to display the file again via the *roff* command. *History* is invoked by typing *!r*. Unfortunately, the user has forgotten that the *rm* command has been subsequently used, and his newly created file is destroyed.

Thirteen expert Unix users were questioned concerning their *history* usage. Of these, only one rated himself as "expert" in knowledge of *history,* while four made full use of all the history functions. The rest invoked subsets of the simpler features, with most not even being aware of *history*'s more sophisticated aspects. In general, those asked indicated satisfaction with *history,* especially in its ability to save time and tediousness of typing. Criticisms usually concerned the painful syntax, lack of documentation, and system limitations and bugs.

History illustrates how certain portions of a modeling task can be done automatically and transparently, while avoiding unpleasant side effects by providing explicit control to the user in other areas. Some existing problems — such as its notation — are design faults, and can probably be overcome at the individual system level. Other areas — such as incorrect invocations due to the imprecise nature of human memory — are perhaps more indigenous to the combination modeling philosophy as a whole.

2.5 Combination Modeling — What, Where and Whence in Videotext

Previous examples highlighted modeling as a system enhancement of the man/machine interface. However, modeling is not merely a refinement; it may also solve serious problems found in the dialogue. This section is concerned with examining *videotext* in this context.

Videotext is a computer system which provides the user a potentially high level of interaction with a data base through a television screen. Users typically select and display pages of information and perform computer-based activities such as electronic mail, electronic shopping and games (Fortin, 1981). These activities are normally accessed via a menu selection dialogue which traverses a hierarchical tree structure, where each menu category represents entries or subdivisions of "fields of knowledge" (Tompa, 1982). Figure 2.1 illustrates four sequential videotext pages, each numbered by the concatenation of menu selections leading to that page. Page 0 represents the first-level menu containing the primary fields of knowledge. Further menu selections narrow the category range to culture (page 3), local entertainment

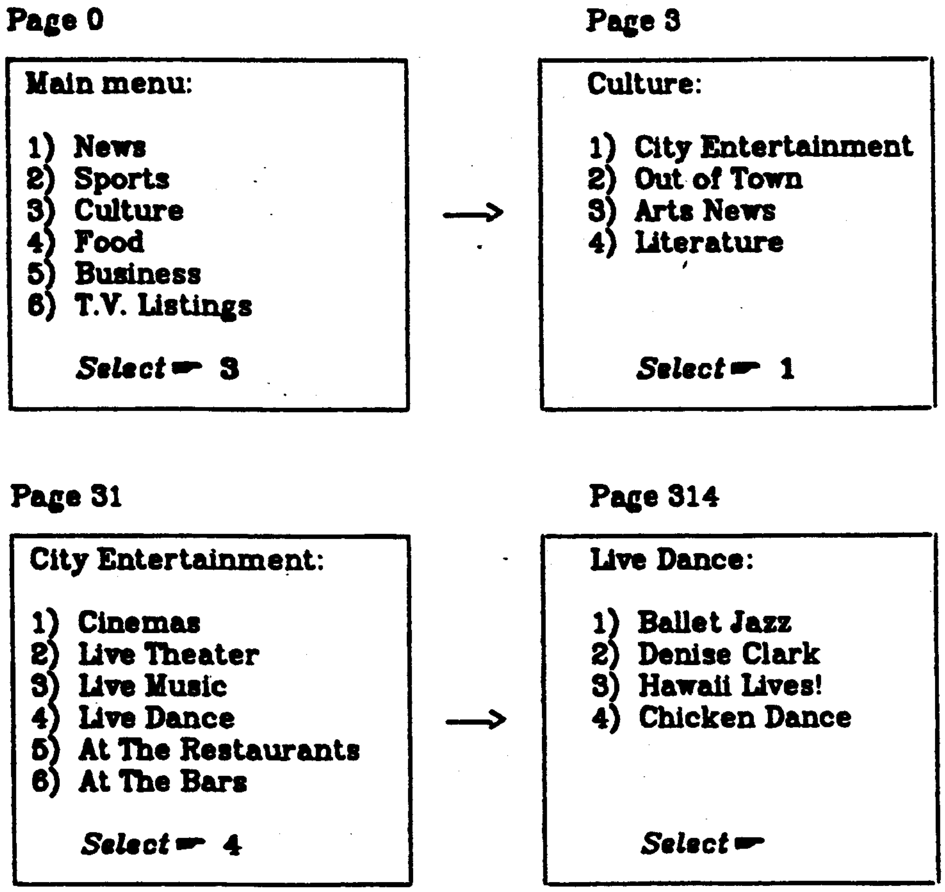

Figure 2.1 Four Sequential Videotext Pages

(page 31), and finally the live dance menu (page 314). A continuation of this process would eventually reveal a data page, such as the location, time, and cost of a particular dance performance. Perhaps electronic booking of reservations is offered. The user may then navigate through another field of knowledge after returning to the primary menu.

Serious drawbacks to this simple menu access scheme are identified by several authors. Selected criticisms are described below.

1) The user may be uncertain about the content of a menu category (Latremouille and Lee, 1981) and will repeatedly probe the hierarchy before retrieving the desired information.
2) Information retrieval may fail for two reasons: the user is unable to select the proper categories, or the information does not exist in the data base.
3) As data bases grow in size, it becomes harder to locate specific items (Martin, 1980). This is particularly disconcerting when one considers that Prestel, a British videotext system, is aiming for one million pages of information by the mid-1980's (Wilkinson, 1980).
4) User confusion and spatial disorientation in a large menu hierarchy will likely occur as a result of the problems just noted (Engel *et al*, 1983).
5) Difficulty in remembering the choice path used to arrive at the current location causes erratic searches on successive probes (Engel *et al*, 1983).

Dumais and Landauer (1982) suggest that some of the menu faults probably arise from inflexibility imposed by most schemes which allow only a single access route to a given item. In addition, system designers invent word meanings and categorization schemes which may not match those of the end user. The latter suggestion is verified in an experimental study of design defects in a menu-driven data base by Whalen and Mason (1981), who found that miscategorization of information was the most serious. Whalen and Latremouille (1981) examined retrieval failure in Telidon menus (see point 2 above) through experimentation. They concluded that people are very likely to stop searching for existing information rather than undertaking extensive searches. Fitter (1979) regards user recollection of where he has been (see point 5) as vital to his feeling of overall system control. Engel *et al* (1983) propose an improvement to data access which minimizes some of the previously mentioned problems. This improvement is *what, where and whence* in videotext.

There are three conceptual components, using two graphics display screens, in *what, where and whence*. The *what* screen presents the primary information the user is attempting to retrieve. The *where* screen displays secondary access information, such as a symbolic textual and graphical map of portions of the data base. These two components attempt to resolve problems one through four listed above, for the user can visualize a greater portion of the information structure. *Whence* deals specifically with problem 5; a history list stores the temporal order of a user's successive choices and touch controls allow him to backtrack at any time. *Whence* will be explored from a combination modeling perspective.

Whence is a temporal model created through implicit and explicit techniques. Each sequential step is recorded automatically. *Whence* also allows explicit page marking, for it recognizes that a user may desire to return to selected key pages. Thus, the system keeps track of the complete access path, while the user indicates the subjectively important pages.

Model elicitation is under complete user control through the use of six touch controls (Figure 2.2). The "FL" and "NFL" buttons allow the user to add or delete respectively a flag from the current page. The symbols "<" and ">" perform single steps backward and forward through the automatically formed history list, while "<<" and ">>" perform multiple steps backward and forward through user-flagged pages.

It is immediately apparent that many of the drawbacks found in the Unix history mechanism are eliminated in *whence*. Confusing syntax is all but eliminated through use of touch controls. Additionally, it is difficult to make catastrophic errors, for the user can simply mark his current location before exploring the history list, guaranteeing a quick return. Flagging of important pages eliminates the need to remember less important in-between pages. The Unix *history* mechanism discussed earlier had no way of flagging important commands, exposing the user to the risk of recalling the incorrect command line.

Of course, trade-offs exist in the clean history mechanism of *whence*. There is a demand on the user to remember the flagged pages, especially if he was indiscriminate in flagging them. In addition, *whence* keeps recording the user's pathway during its own invocation. Engel *et al* (1983) report that the corresponding new piece of history does not fit the user's notion of the actions performed. This problem is practically eliminated when the recording of page numbers during history consultation is inhibited.

Whence addresses the serious human factor problem of a user knowing where he has been during a videotext access. Engel *et al* suggest that this is close to daily life practices such as marking pages in books through dog-earing or book-marks. User confidence and control is increased by allowing him to flag and return to a known node in the hierarchy when exploring an uncertain branch.

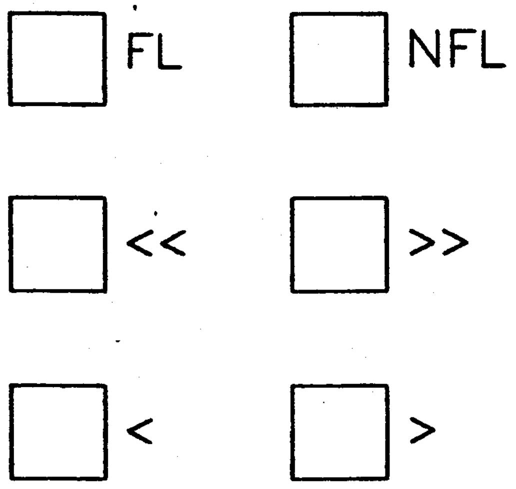

Figure 2.2 Whence touch controls

3. Why Model Users? — A Selected Literature Review

It is hard to utter common notions in an individual way.

(Horace, 65 — 8 B.C.)

Rules and models destroy genius and art.

(Hazlitt, circa 1800)

Human factor studies in interactive systems form a relatively new topic in computer science journals. The early "landmark" papers, e.g. Hansen (1971), Foley and Wallace (1974), were written mainly within the last decade. The notion of user modeling is even more recent. It was not until 1982 that a journal such as the *International Journal of Man-Machine Studies* devoted part of an issue to this field. A researcher must therefore be content at present with piecing together isolated articles, asides and related ideas; for very few concentrated studies exist.

The current literature is divided into two areas: general overviews and implementation-specific discoveries. Implementation papers normally concentrate on the techniques, usually dealing with issues arising after the fact. Often, the issues are inextricable from the implementation and are thus ill-suited for inclusion within this chapter. The general overviews attack the basic problems of user modeling, although very few address adaptive systems directly. It is this general literature which is of interest here, for the fundamental issues are addressed. Unfortunately, it usually presents a series of personal views, intuitive feelings, folklore guidelines, and implementation ideas rather than concrete data formed from tested hypotheses. A designer seeking guidance from these authors finds himself immersed in a quagmire of platitudes and contradictions. This chapter will summarize what was said by different authors in an attempt to highlight the relevant topics and issues in user modeling.

3.1 The Desirability of Modeling

There are two tragedies in life. One is not to get your heart's desire. The other is to get it.

(George Bernard Shaw, 1856 — 1950)

Many authors imply or state outright that system personalization at some level is desirable. Primary reasons supplied by these authors to justify user modeling are: variations in user experience, evolving user needs, dialogue determination and user/designer conflicts. Each of these areas is studied in turn in the following sections.

3.1.1 Variations in user experience

It was previously stated that different canonical models are appropriate for users with different levels of experience. Unfortunately, it is often difficult to fit a system into a particular category. Edmonds (1982) writes that users of the same system typically will include both novices and experts, a concern echoed by other authors. Larson (1982), in an introduction to end-user facilities, mentions adaptability as a factor affecting the end-user's perception of a computer system. He asks:

> Does the system adjust to the end user's level of competence as he becomes more experienced? Does the system tailor itself to the habits and styles of different users?

James (1980) proposes certain desirable qualities of the user interface, the second of which is adaptability.

> We are concerned here particularly with the ability of the computer system to provide a dialogue which is consistent with the user's previous experience. This implies that the system is able to learn a "required" pattern of behavior from a particular user and also to remember and reintroduce this pattern when the same user is in touch again. There is no way in which a system with a permanently fixed "level" of response can satisfy a group of users with a very wide range of experience.

All three authors are concerned with variations in user experience, and stress the need for systems that can adjust or be adjusted to a wide range of user ability.

3.1.2 Evolving user needs

Some authors are concerned with the evolving nature of user needs as opposed to the unusually high inertia associated with modifying a computer system. Edmonds (1982) proposes this argument for adaptive design in man-computer interfaces:

> ... in a changing world we need to ensure that a system working at a given time should satisfy the needs of that time rather than those identified at some earlier stage.

Innocent (1982) supports the idea of the dynamic user even more vigorously when he writes:

> The problem for a designer is that the user is likely to change both within and between interactive sessions with a computer. Hence, there is a need for a modification of the model at different times.

Eason and Damodaran (1979) expand this idea in a paper discussing procedures for including user involvement in the design process. Although the arguments are directed towards flexibility in the canonical modeling domain, they apply to all user modeling aspects:

> The second purpose of user support is to promote continued viability of human/computer interaction. This evolutionary user support function is complex and varied since it is concerned with an evolving computer system applied in a changing environment through the activities of a human user who is also changing as

> a result of his learning from experience. The importance of supporting an evolving human/computer relationship lies in the fact that unless a given system can continue to develop and to match the needs of the environment in which it functions, it ceases to have value. It is therefore essential that the system develops in a way which is consistent with development of the users' interests, needs and skills.

Maguire (1982), in an evaluation of previously published recommendations of man/computer dialogues, suggests that "the lifespan of the system can be lengthened by designing it to be adaptable to the user's evolving needs".

Two main points are brought out by the above authors. First, it is not enough to recognize that different levels of user expertise are contained in the population. Each user can also drift from category to category as his experience evolves. Although this evolution is usually progressive, a regression of ability may also occur when the system has not been used for a long interval. Secondly, any change in the application task will alter the user demands on the system. Not only may the application needs change globally, but individual users will also have different local goals at different times. A rigidly designed system would be ill-equipped to handle the new demands, thus leading to user frustration due to the complexity of working around the system constraints.

3.1.3 Dialogue determination

Thimbleby (1980) introduces the term ***dialogue determination***, which is defined as the degree of control expressed between the system and user. A dialogue is over-determined if the computer overly restricts the dialogue and under-determined if the user is at a loss as how to accomplish a task. Thimbleby concludes:

> Whether a user is under- or over-determined depends on his experience and frame of mind as well as the implementation. The same dialogue may be under-determined for one user and over-determined for another user, or even the same user at another time: an ideal computer system should adapt itself (or should have been adapted in the design stages) to the specific needs of the users, even on a day-to-day basis. Perhaps the panacea ... is for users to have personal and idiosyncratic interfaces.

Dialogue determination is often supplied as a characteristic of the user population. Cuff (1980) recommends that a dialogue should offer the casual user

> ... a "constrained choice" interface which exposes a relatively small number of items to consider at any one time, and implicitly or explicitly guides the user to a solution.

Expert users would probably resent this constrained dialogue. Shneiderman (1980) notes that experienced users should have especially powerful commands available.

The distinction between power and control in determination is subtle. A satisfying system gives the user the sense that he is in control (Shneiderman, 1980). Yet different users will be supplied with different constraint levels to achieve this goal. It can be argued that the high variation in dialogue determination levels across the user population is best dealt with in an adaptive manner.

3.1.4 User/designer conflicts

An additional criticism of non-adapting systems is the possibility of conflicting views between user and designer. James (1980) provides this reason for the growing dissatisfaction with the performance of existing computing systems:

> ... the systems do not do what is required by the users; there is a mismatch between the views of the designers and the users as to what was originally required of the system.

Groholt, a highly experienced systems manager, is quoted in Eason and Damodaran (1979) as saying that

> ... user involvement (in design) has rarely been successful because neither designers nor users have access to procedures suitable to this purpose.

One can presumably solve the problem of user/designer conflicts by providing proper tools for the interaction. However, even this may have its faults, for the consensus of the many does not necessarily provide an ideal interface. In addition, a good interface based on partially unpredictable user behavior implies that designers need sophisticated models of users (Innocent, 1982). There is no guarantee that this model can be successfully fashioned by the user/designer team.

In summary, successful interaction between user and designer is inhibited by conflicting views, lack of dialogue tools and inadequacies of constructed models. The adaptive interface may provide the means of minimizing the conflicts that arise in even the best designer/user system, for the model will be constantly updated and corrected.

3.2 Criticisms of User Modeling

The injustice done to an individual is sometimes of service to the public.
(Junius, circa 1770)

It has been shown that many authors see user modeling in adaptive systems as highly desirable. But it is too simple to present modeling as a solution to a great many ills, for it introduces associated complexities. Few authors have actually explored adaptive systems to any great depth, but those who have bring up a selection of inherent problems in the following areas: dynamics of user/system concurrent modeling, difficulty of implementation, evaluation of benefits, questions about dialogue determination, and inaccuracy of model construction. Each of these areas are studied in turn.

3.2.1 Dynamics of user/system concurrent modeling

This thesis deals with the possibility of the computer fashioning a model of the user for adaptive purposes. However, this model is not being created in isolation. Gaines and Shaw (1983) emphasize that the user will also create a model of the system:

> People automatically form internal models of their environment [for] the mind is all the time searching out patterns of cause and effect. The models formed by users may vary widely and have many connotations that arise from their previous background experience and bear no direct relation to the system.

Edmonds (1982) supports this idea when he proposes a definition of the interface as the part of the system that represents the user's model of it. However, he then mentions:

> A problem with this view is that different users may have different models of the system.

The concurrent effort by the system and the user to create models of each other leads to inherent difficulties. Gaines and Shaw (1983) express concerns over the programming of adaptation to the user within the system:

> ... the user is not only coming to comprehend the system but [is] also himself adapting to it. We have a control-theoretic situation in which two coupled systems are each attempting to adapt to one another and instability may result.

The instability of concurrent modeling is elaborated on by Innocent (1982):

> Stability of the [interface] facade may interact with the stability of the user in terms of knowledge and experience in such a way that the total man/computer system is unstable. That is, each change to the facade may upset the confidence of the user who expects a facade to be consistent and uniform rather than adaptive and changeable. The balance between the need for a consistent and uniform facade and the need for change to suit user, task and system environments is a delicate one. Little is known about the factors affecting this balance and how it affects the ease of use and learning of an interactive system.

Concurrent modeling is extremely difficult to design around, for it is almost impossible to predict the consequences. However, there is no doubt that people do have the capability to react favorably to changing circumstances in everyday life. Successful adaptive systems will probably be those that create an environment in which the changes that do occur are in themselves perceived as predictable and rational.

3.2.2 Difficulty of implementation

Simple systems are easier to implement than complex systems. Thus, designing adaptive techniques into canonical systems adds a new level of implementation complexity, at least in the short term. Eason and Demoderan (1979) state:

> It is clear that defining design criteria for evolutionary support mechanisms is far more difficult than describing minimum user support needs.

Gaines and Shaw (1983) cast doubt on the feasibility of automatic adaptive systems:

> ... the channel of communication from person to computer just does not provide a sufficient flow of communication for on-line identification of the characteristics of the person. ... It is possible to gather specific useful information about individual users if this is designed into the original system, but the time-constant of doing this will be too long for immediate action within the interactive dialogue.

It is difficult to accept or refute these statements due to the absence of hard data on adaptive systems. Measures are needed to explore the above issues, such as the long term cost of implementation versus the supposed benefits supplied to the user. Case studies of adaptive systems must be made to demonstrate their feasibility.

3.2.3 Evaluation of benefits

An adaptive system must somehow be evaluated to verify that benefits are actually being supplied to the user. It is not clear how to accomplish this evaluation. Innocent (1982) writes:

> ... there is a major problem in evaluating interactive systems, in that suitable unobtrusive methods based on sound statistical principles have to be developed.

This becomes especially complex when the system is constantly changing, for sound experimentation should have as few uncontrolled variables as possible. Shneiderman (1980) is a strong proponent of experimental methods. Even so, he concedes that software quality measurement is an infant discipline.

> As this speciality matures, the ability to measure will develop, but in its youthful phase, there are conflicting opinions as to what and how software characteristics should be measured.

It is probable that much work based on very simple adaptive systems will have to be done to create the foundation of principles necessary for complex system analysis. As Shneiderman (1980) adds:

> Each small result is like a tile in a mosaic: a small fragment with clearly discernible color and shape which contributes to the overall image of programming behavior.

3.2.4 Dialogue determination revisited

It was mentioned previously that dialogue determination is the amount of control expressed between the system and the user. Gaines and Shaw (1983) warn that, at the present state of the art, the user should dominate the computer. This conflicts with Thimbleby (1980) who indicates that the level of determination is highly dependent on "a user's experience and frame of mind as well as the implementation".

Explicit modeling provides the user with a high degree of control over certain dialogue aspects, a control which may be too under-determined for the inexperienced user. This same control may frustrate the experienced user due to the tediousness of altering the model for short-term task needs. Automatic modeling may reverse these problems, for it implies that the system has greater control. If the automated system prescribes an action which is beyond the understanding of the user, then he will probably experience a loss of dialogue control. Perhaps a prime application of user modeling is the ***adjustment*** of the determination level during dialogue progression, especially in considering the diverse user experience and their evolving needs.

Determination is ingrained in user modeling. Although there is controversy in how it should be applied, it is likely that modeling will be used to tailor a determination level to a user. Again, human factor experiments must be carried out, based on well thought-out designs, to test the dynamics of user/system control.

3.2.5 Inaccuracies of model construction

Automatic modeling attempts to construct a model by analyzing the user input supplied through the interface. However, there is a danger that the model that is being built is inaccurate. Rich (1983) warns:

> The thing that all of these [modeling] techniques have in common is that they involve guesses about the user. These guesses are made by the system on the basis of its interaction with the user. As a result, the possibility of error must always be considered.

Cuff (1980) identified ***feedback*** as a two way interchange; not only is the system providing feedback to the user, but the user is also providing feedback to the system:

> ... it may be the ***system*** which derives such benefit from the ***user's*** actions. This is particularly the case for a system with a strong heuristic or model-building element, where its attempt at resolving an inexact query is based on the uncertain model of the questioner which it has at any given time. The user's response to output resulting from this attempt can strengthen or correct the developing model.

Both Cuff (1980) and Rich (1983) agree that models generated by the system may be inaccurate. An effective system must therefore constantly re-evaluate the probability of correctness of model aspects and discard any of low probability.

3.3 Conclusions

Are adaptive systems a viable alternative to static ones? Clearly, this chapter has not answered this question. The issues involved are many and do not indicate whether or not user modeling would benefit users. The remainder of this thesis will concentrate on defining and exploring an application of interactive computer systems which is amenable to automatic user modeling. Emphasis is on demonstrating that such applications actually exist, and in suggesting specific guidelines for a test system. This test system is then described and implemented. Afterwards, a pilot study is undertaken to investigate the effectiveness of the adaptive system component.

4. Repetitively Accessed Data Bases

The personality [an interface] should attempt to project is that of a ***servant*** — a very Dickensian one with wit and intelligence rather than servility ... dedicated to his master's ends.

(Brian Gaines, 1981)

The ideal personalized system is reactive to a user's needs in much the same way as a Dickensian servant would be. This "servant" can be envisaged as a compassionate "mentor"/"secretary" composite. The user is the task orchestrator, and his actions are guided by the "mentor". The "secretary", on the other hand, removes the burden of tedious and repetitive tasks while predicting and preparing for the user's next action. Unfortunately, this ideal composite "servant" is not yet available in computer systems. Artificial intelligence is still coming to grips with the knowledge-based expert systems fundamental to the mentor described. Certainly, immediate real-time response is not yet possible even with the best expert systems currently available. But office automation tools do exist for some secretarial functions. Word processors, spell and style packages, integrated office systems such as Apple's Lisa (Williams, 1983), and a myriad of other specialized tools are flooding the market. However, most do little more than allow the user to become a "super-secretary", as he is still responsible for setting up and executing a desired action. The general adaptive system is still in the future, for all the individual components have yet to be proposed and analyzed. But a truly reactive system can be built and studied for specific functions, such as reducing repetition in the user's task.

This chapter examines the premise that specific data bases are repeatedly accessed for the same information, and that general access characteristics exist. Guidelines for the design of a personalized access language which removes task redundancy can then be extracted from a study of users' access patterns. The first section briefly surveys existing data bases and their access languages, and then supplies a definition of a repetitively accessed data base. The second section examines, extracts, and formulates characteristics of the repetitively accessed data base. Finally, guidelines for the design of a reactive data base access language are proposed.

4.1. The Data Base Spectrum

The term ***data base*** has no precisely defined meaning. Bradley (1982) defines a data base as "a collection of cross-referenced files", a description which would exclude any file systems that did not cross reference each other. Date (1977) describes it as "a collection of stored operational data used by the application systems of some particular enterprise", a meaning with connotations of data bases falling solely in a direct application domain. Both authors go on to describe the traditional data bases and their query languages — hierarchical, network or relational architecture accessed via Codasyl, SQL, DSL-alpha, Query-by-example and others.

Unfortunately, the definitions and system illustrations given by the above authors neglect very simple data bases and file systems that are becoming increasingly useful. For example, ***help*** facilities are now standard suggestions in virtually all human factors recommendations for man-machine dialogues (Maguire, 1982; Gaines, 1981). A typical ***help*** system may display a series of selected text files describing a command. However, this information system cannot traditionally be considered a data base — it is disallowed by the first definition for it is not cross referenced, and the second excludes it for being a support package rather than an application system of an enterprise. Traditional nomenclature would label ***help*** as a file system rather than a data base. However, there are systems which do not fall cleanly into either category. ***Browse***, created by Bramwell (1983), adds to a ***help***-like file system cross-referencing capabilities with minimal change to the file contents. The gray area separating a data base from a file system and the increasing public usage of the term data base to cover any and all file systems makes it necessary to clarify the nomenclature. For the purposes of this thesis, the definition of a data base is expanded to include file systems that provide information to users via simple access languages.

Data bases, their query languages and their users are highly interrelated. Some query languages are oriented toward end-users — the ones making the original query — while the complexity of other languages necessitate an intermediary, that is, a middle man between the machine and the end user[1]. DSL-alpha is a high level non-navigational language which allows complex retrievals from a relational data base (Bradley, 1982). Its users are expected to be intermediaries, for the retrieval language is solidly based upon set theoretic expressions. Codasyl, on the other hand, is a low-level navigational language which based upon a network schema. Its users are expected to be system specialists with a good knowledge of how the data base is actually set up. The ***help*** system previously mentioned may use a very simple access language, such as a keyword or menu indication to retrieve a text file of information. Such a system would concern itself primarily with naive to expert end-users. Videotex systems, such as Prestel in Great Britain, provide the general public with access to extensive data bases. The access language is typically through simple hierarchical menu selections in which the menu hierarchy maps directly on to the data base architecture[2].

The brief survey described above indicates that the complexity of a query language usually increases with the organizational complexity of the data base. With the emphasis in computer systems shifting toward the casual user, as illustrated by the proliferation of personal computers, some means must be taken to allow him to gain easy access to data base contents. Automated tellers are a striking example of how totally untrained users can successfully interact with a highly complex data base, albeit for limited purposes.

The data base spectrum has been shown to cover a broad range. Analysis of human factors and suggestions for improvements in information retrieval must therefore be to a specific type of data base, its access language, and its intended user. To do otherwise would give results too general for real application.

[1]A brief literature survey of the role of an intermediary is found in Martin (1980).
[2]An implementation of such a hierarchy in Telidon is found in Abell (1981).

4.1.1 A definition of repetitively accessed data bases

A repetitively accessed data base (*rad*) is defined as a database in which most users predominately access items that they have accessed before; that is, they exhibit a repetitive pattern of accesses. The actual set of items retrieved may be disjoint, overlapping or identical for different users. The frequency of repeated accesses may also exhibit high variation across users, provided that the average frequency is fairly high.

Examples of *rad*'s in the non-computer world can be found. A cookbook has a subset of recipes referred to repeatedly by a single homemaker. However, usage patterns differ, as not all people favor the same recipes. Some cooks prefer tried and true recipes, and will thus use a small set of recipes many times. Others desire variety and thus use a large recipe set with little repetition. A similar analogy may be made to a book of verse, readings in the bible or reference manuals. Dumais and Landauer (1982) report from M.E. Lesk's analysis of work logs that up to 70% of the time Boeing engineers had looked for specific things that they had seen before but had forgotten (e.g. standards, product manuals). Explicit and implicit modeling also have their loose analogies in the non-computer context. A cook can explicitly mark his recipes by using bookmarks, whereas implicit modeling takes place by the book naturally opening to highly used locations through wear of the binding.

Rad's are specifically defined as data bases where most users exhibit a high repetition rate of queries. The time period can vary according to the usage patterns of the data base. For example, a very short-term task-intensive data base may have a time period of one login session. Yet another data base which exhibits long term usage patterns may have a time period equal to the life of the user on the system. Let us define a ***unique access*** as the retrieval of a data base entry that the user has not previously retrieved. A ***repetitive access*** is the retrieval of a data base entry that the user has previously retrieved. ***Total accesses*** are the total number of accesses, unique or otherwise, that the user has made to the data base. A ***unique access rate*** can be denoted as:

$$\textit{unique access rate } (\%) = \frac{\textit{unique accesses}}{\textit{total accesses}} \times 100\%.$$

The ***repetitive rate*** can now be given as:

$$\textit{repetitive rate } (\%) = 100 - \textit{unique access rate}.$$

Rad's promote potential for automatic user modeling because there is opportunity for the query language to give preferential treatment to a repetitive access. An example is replacing a static hierarchical menu retrieval system with a dynamic one which places previously retrieved items high up on the tree, thereby reducing the number of menus the user must go through to retrieve the desired item. A complete description of a system of this type is described in Chapter Five.

Simple personalization schemes may alleviate specific data base retrieval problems. Some of these schemes are based on the assumption that a data base exhibits a *rad* profile. However, the existence of *rad*'s in the computer world has not been discussed in the literature. Therefore a case study of a data base suspected of manifesting the *rad* property has been undertaken and will be described next.

4.2 A Case Study of the Unix *Man* System

Statistical analysis of a suitable data base is needed to support the existence of ***rad***'s as a subset of the data base spectrum. The data base to be examined must fulfill certain requirements for this analysis:

- Ease of information extraction by the data base user is necessary. Complex access mechanisms may introduce variables that can confound results.
- The data base should be suspected of having a ***rad*** profile. As this study aims to establish the existence of ***rad's***, biasing the data base choice will facilitate the search process.
- It must allow collection of enough raw data over a large population for a sufficient time period to provide significant results.

The Unix ***man*** on-line manual system was inspected as a potential candidate. A brief description of ***man*** is given by Bramwell (1983).

> Unix supplies an on-line documentation system which uses a program called ***man.*** The name is a contracted form of the word "manual". This allows the user to access text files which describe commands, system calls, subroutine libraries etc. The level of information provided is aimed at users with some computing expertise who wish to make use of the various operating system facilities. Since technical details of the interfaces to such facilities are often required on very short notice and are frequently forgotten, an on-line manual is an ideal way to provide this information.

Man was felt to adequately fulfill all the aforementioned requirements. First, the access mechanism is straightforward, The user simply types ***man <keyword>***, where the keyword is the command to be described. Secondly, retrieval is likely to be repetitive, for a user will often forget the precise invocations of even frequently used commands. Finally, the university environment supplies a large user population and relaxed privacy restrictions to allow adequate raw data collection.

4.2.1. Previous studies of man usage

A general analysis of data collected over a two month period on the same subject group has been done by Bramwell (1983). He found that:

- Users were predominantly first year students and thus novices to the Unix system.
- The average ***man*** user had an overall failure rate of 25% in ***man*** uses, due to a variety of known and unknown reasons. No difference was found between failure rates in naive or expert Unix users.

4.2.2. Data collection and analysis — a population study

The following study is based on raw data of *man* usage collected over one month — from February 25 to March 24, 1983. Users were mainly computer science undergraduate and graduate students, research assistants and professors. Out of a potential user population of 1294, 822 actively called on the *man* data base, with only 443 able to get at least one valid retrieval. A total of 4978 correct *man* retrievals was made.

Man usage by single users was surveyed and compiled. Table 4.1 combines the number of correct total accesses in *man* with the average repetitive rate found for that range. Unique accesses and the unique access rate are supplied as intermediate results. Figure 4.1 is a graphical representation of the results in the table. The dotted line represents the 50% repetition level. The tabulated data supports the following results. Firstly, the repetitive rate increases with greater man usage. Figure 4.1 graphically illustrates that the low 13% repetition found with less than five *man* uses quickly approaches 50% when more than ten accesses are made. Secondly, many repeat accesses are made by each user on a small subset of the *man* data base. For example, Table 4.1 shows that for 31 — 50 man uses there was a 51% repetitive rate on an average of approximately 19 unique accesses, a very small subset of *man*'s 426 available selections.

4.2.3. Data analysis — a sample study

Number Correct Accesses	Unique Accesses (Average)	Unique Rate (%) (Average)	Repetitive Rate(%) (Average)
1– 5	1.8	86.5	13.5
6–10	4.9	65.6	34.4
11–15	7.7	57.7	42.3
16–20	9.4	51.3	48.7
21–30	13.3	53.4	46.6
31–50	18.6	48.7	51.3
>50	35.9	47.0	53.0
ALL	6.0	72.5	27.5

Table 4.1: Single user averages of *man* usage

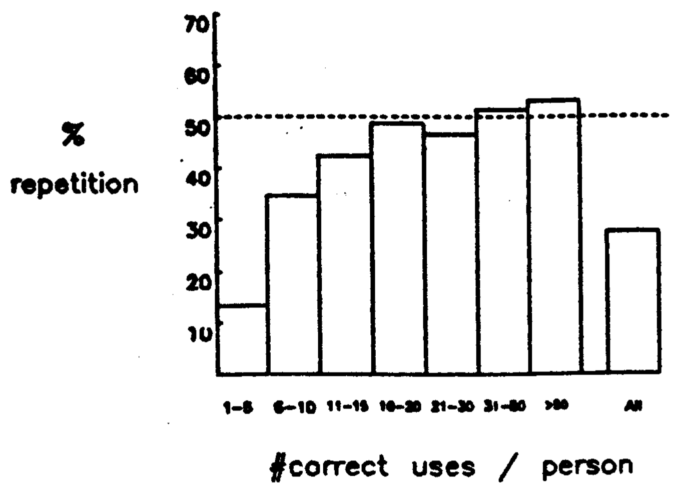

Figure 4.1: % Repetition vs Correct uses / person

It is not possible to get a complete picture of *man* usage through the above statistics. Unanswered questions remain, such as:

- What is the degree of variation present in the repetitive rate between users? The *rad* definition provided in section 4.1.1 demanded a high rate, as is shown in *man* usage. However, it would be interesting to examine whether or not a high or a low variation in the average repetitive rate was exhibited across users.
- Are a single user's repetitive accesses uniformly distributed over his total entries, or are some entries accessed much more frequently than others?
- Each user uses only a small subset of the *man* entries available to him. Are these subsets disjoint, overlapping or identical across users?

These questions are partially answered by examining a particular group of users: those who had completed 17 — 19 valid selections. The criteria for selecting this sample are that users should have retrieved enough entries to be familiar with the system and that the sample should be large enough for adequate study. The repetitive rate was looked at in detail, giving results tabulated in Table 4.2 and graphed in Figure 4.2. The histogram is used to illustrate the distribution of repetitive rates across all users. The dotted lines provide a benchmark: they represent the 50% range of the sample users closest to the mean. A high degree of variation in the repetitive rate across users is shown numerically in Table 4.2 and graphically in Figure 4.2. The range, which describes the difference between the users with the highest and lowest rate, covers almost 60% of the possible spectrum. A closer look at the histogram reveals that about 50% of the sample had a repetitive rate within a 10% range of each other, a high contrast to the overall range. Although this indicates that standard repetition patterns do exist, the wide variation noted over the complete sample cautions against stereotyping this access profile across all users.

Measure	Result
Average Repetitive Rate	45.2%
Maximum Repetitive Rate	70.6%
Minimum Repetitive Rate	11.8%
Range (Max - Min)	58.8%
Standard Deviation	17.1

Table 4.2: Repetitive rate analysis for users with 17-19 valid selections

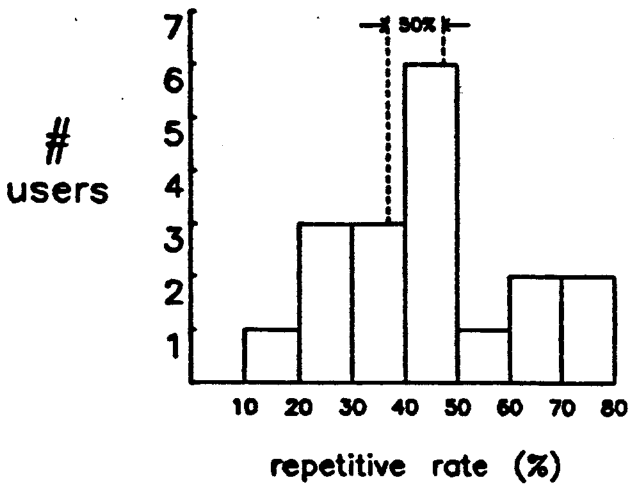

Figure 4.2: Distribution of repetitive rates across users

The same sample group was used to investigate commonality of command access, that is, whether or not the subsets of keywords subjects used were disjoint, overlapping or identical. All keywords accessed by the sample were tabulated against the actual number of users who had accessed them, with the results plotted in a histogram (see Figure 4.3). A total of 86 unique entries — 20% of the possible 426 available entries — were asked for. Of these, 58% were requested by one user, 19% by two, 12% by 3 , and only 11% by 4 or more users. This was a rather surprising result, for many of the users were students in the same undergraduate courses with identical programming assignments to solve and, one would assume, similar needs. The fact that only minimal commonality of command access was found implies that users had different needs even for similar tasks.

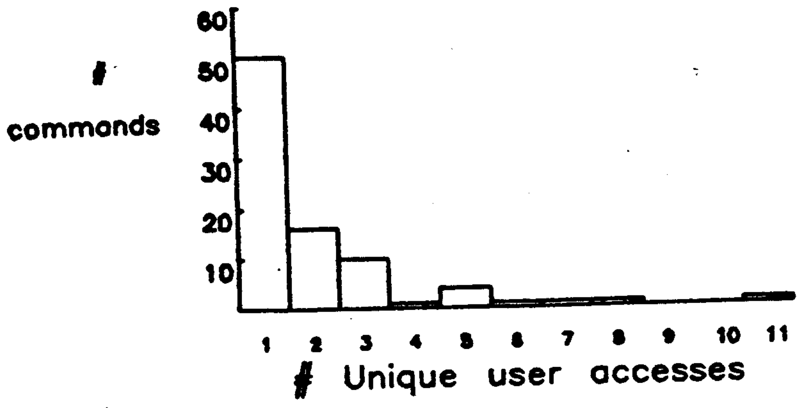

Figure 4.3: Commonality of Command Access

A single user may exhibit a variety of patterns in distributing his repetitions over the entries he makes. An example is a user with a total of 20 accesses on 10 keywords and a 50% rate of repetition. Many distributions can provide these results. Each keyword may have been accessed twice, 9 keywords may have been accessed uniquely with the remaining one accessed 11 times, and so on. Scores within one standard deviation (66%) of the sample population were studied qualitatively in an attempt to discover if any particular distribution patterns predominated. It was found that only a slight pattern existed. Some users had very high repetitive rates on relatively few entries while others had many entries occurring less frequently. In general, for each user there was always a portion of the entries which had only one access, another portion with 2 or 3 accesses performed on each, and a small number with a relatively high repeat frequency. It must be stressed that there was enough variation away from this pattern to negate any stereotyping of a typical user.

4.3. Conclusions and Suggestions

The Unix *man* system was studied in order to examine repetitive access patterns by individual users. A variety of aspects were investigated, providing the following results:

1) The repetitive rate was found to be very high, approaching 50% after relatively few retrievals. *Man* can therefore qualify as a *rad* system as defined in section 4.1.1.
2) Only a very small subset of the data base was accessed by any single user.
3) Marginal commonality was found between user data base subsets, even when user tasks were similar.
4) The majority of users have similar repetitive rates. However, there was significant variation away from the norm.
5) The distribution of access repetitions over entries varied substantially from user to user. In an individual user, no uniform distribution was found — rather, the frequency of access per entry also varied substantially. However, the general trend was to access most keywords

between one to three times with a smaller set being called on at a higher frequency.

In general, one can conclude that access in *man* is highly repetitive, and although it is possible to envisage the characteristics of a typical user, it would be a mistake to create a neat stereotype.

The original intention of this study was to discover the existence of a *rad* system and to analyze its characteristics and apply them to the design of personalized access techniques. These techniques must consider almost all users, for a well designed system should be able to accommodate 95% of its users, a range that was shown here to contain a high degree of variation. With this in mind, the above results can now be used to suggest loose "guidelines" that a personalized access scheme must meet. It should be stressed that these guidelines are not absolute — they are merely better than having no guidelines at all.

1) Access to repeated items must be treated preferentially in terms of ease of access. (*50% of accesses are repeats.*)
2) First time access to entries must not be noticeably longer than in the unpersonalized system. (*50% of accesses are unique.*)
3) Personalization should be done at an individual, rather than system or group level. There is not enough similarity between users to justify the group approach for easing accesses over items that show higher frequency usage on the whole. (*Marginal commonality between user subsets.*)
4) Personalization schemes must be able to handle a wide variety of usage patterns, be very advantageous to users with medium to high repetitive rates, and yet transparent and/or beneficial to individuals with low rates. (*A significant number of users differ from the norm.*)
5) The system should be primed to a sensible default state to take advantage of any group patterns that do exist. However, the worst case use of this state should not differ in access speed from an unprimed state. (*A marginal commonality does exist.*)
6) Individual items with different access frequencies should be graded according to that rate. An oft-repeated item should have preferential treatment in terms of ease of access by the user than a less frequently accessed one. (*No uniform access rate for all keywords exists; patterns vary substantially.*)

A *rad* system was discovered and analyzed, and suggestions for personalization were made based on the results. A system incorporating these recommendations will now be described as an alternative to traditional data base access, thus allowing a human factors pilot study to be carried out. The study is necessary to test and discover human factor problems in a personalized system as opposed to a non-personalized control. The success or failure of the tests will allow some verification of the proposed guidelines.

5 A Case Study in Automatic User Modeling

Well, if I called the wrong number, why did you answer the phone?
(James Thurber, 1894 - 1961)

This chapter examines the effectiveness of automatic user modeling in facilitating transactions with computer-driven telephone auto-dialers. The first section reports a brief study of the characteristics of normal telephone use, with emphasis on discovering whether telephone number access follows a *rad* profile. Previous studies and existing autodialers are also noted.

The second section examines a menu-driven selection process for retrieving entries from a large ordered list such as a telephone directory. Menu-based schemes for retrieving items from the directory are devised and assessed. A key feature of these schemes is that they automatically store information about accesses which have been made and so construct a user profile. This profile is used to provide rapid access to popular entries by minimizing the number of menu pages that must be scanned to retrieve these items. One important aspect of the scheme is that it affords the possibility of a device-independent menu-selection system. To support the developing user model, menu pages must be computed on demand from the underlying database, rather than being stored explicitly as they are in most menu systems. The computation can therefore take into account the size of the display and selection device at enquiry time — an ability that is not possible with pre-stored menu pages. Moreover, the ability to work with M-way selections for arbitrary M allows for the widely differing needs of handicapped people (Raitzer et al, 1976) to be accommodated easily and naturally. Additionally, the work seems to be particularly timely in view of current trends towards economizing information processing. Storage has become a cheap resource, but the cost of producing and maintaining individual databases is still high and is increasing. The price of store filled with data is completely dominated by the expense of preparing that data. Mass-marketing techniques, which amortize the development of a resource over its many users, are already beginning to appear in the form of consumer information systems and current-affairs databases. It seems probable that methods for personalizing cheap, mass-marketed databases will have a major role to play in the future development of information technology — especially if these methods are automatic and require no explicit action by the user.

The aspect of this study which makes it particularly timely and relevant to current research problems in human-computer interaction is its provision of a simple and explicit example of automatic user modeling. The extreme simplicity of the directory task — retrieving items from an ordered list — makes it possible to study the user modeling problem in a lifelike situation which is uncluttered by unnecessary detail. Simple human factor pilot studies, to be described in the third section, can then indicate if automatic user modeling is a help or a hindrance.

A disclaimer must be made — the personalized directory system described here is not meant to be ready for marketing. The reader must remember that it is not the intent of this thesis to produce a production system. Rather, it is to create a simple but realistic vehicle for applying the suggested *rad* guidelines and to test the effectiveness of the automatic adaptive system.

5.1 Telephone Usage — A Limited Study

The telephone is one of the most frequently used technological instruments in western society. It is rare to find a home without one. What is surprising is the relatively primitive means the public has of accessing the complex switching networks through the dialing of numbers. It is only recently that micro-computerized auto-dialers — called "teleterminals" by some — have appeared on the market. However, even these are highly limited. Most merely allow the user to soft-wire a few telephone numbers to buttons, which is then given a mnemonic — such as a person's or a firm's name — for identification. This limited and explicit personalization can certainly be improved upon.

5.1.1 Previous studies and existing units

Current work in teleterminals is minimal and unimaginative, in sharp contrast to the market potential. Most units either provide full computer facilities, such as qwerty keyboards and secondary data base access, or highly limited services which offer few advantages over normal telephones. Bell Labs initiated experiments in 1976 in telephone/computer hybrids (Hagelbarger and Thompson, 1983). The first unit had very primitive functions which included a limited fifty-five character display and a full keyboard. The second prototype resembled a business telephone, with a normal touch-tone keyboard and ten electronically-labeled "function keys", which selected and auto-dialed the chosen number.

The latest Bell Labs experiment is a Unix-based prototype offering menu trees to both system and "telephony" functions (Hagelbarger and Thompson, 1983). For example, the root menu may offer access to general and personalized directories, specific phone numbers, and the Unix operating system. Special functions, such as the mailing of memos to other teleterminals when no answer is received, are also included. Personalization is explicit; the user changes the tree himself to reflect his desires. Popular and unpopular items may be shifted around the tree, which is then "recompiled". User reaction to this teleterminal system was mixed, although generally favorable. The reported problems are similar to some of those found in large videotext hierarchies, as described in section 2.5. With hindsight, the other drawback in the Bell prototype is the extra workload and inconvenience to the user who wishes to personalize the system. The alteration and recompilation of trees discourages temporary relocation of popular items up and down the tree.

When one moves out of the lab and into the marketplace, the picture is bleaker still. A recent *Telecommunication* magazine (July, 1983) featured two teleterminal products. The Tel Terminal, built by Digital Transactions, offers a small four-line liquid crystal display screen, with its main telephony feature employing a two hundred and fifty entry "speed-dial" directory. The Davox Deskset (Davox Communication Corporation) presents a full screen display, with a smaller eighty-five name directory. As with the experimental Bell teleterminal, both systems require explicit and primitive personalization.

Clearly, there is much room for development of teleterminals. The current approach seems to create a piece of hardware, and then find out if it fits the user's needs. This approach will be reversed in the following section; user habits concerning telephones will be explored and a system proposed based on the findings.

5.1.2 A sample study of selected telephone users

A serious problem facing the designer of a sophisticated personalized teleterminal is the collection of customer data. Telephone usage, rental and sales is highly competitive, with companies becoming reluctant to share any collected data for fear of its piracy by a competitor. Caution is also necessary to avoid violation of privacy laws. Attempts made to gain hard data on individual telephone use from Alberta Government Telephones (AGT) for this thesis were futile; the information either did not exist or the administration was unwilling to distribute it. It thus became necessary to conduct a limited study of individual telephone usage for the specific purpose of this thesis.

Fourteen telephone users known to the researcher were asked to keep a list of all calls originating from their office and/or home telephones. Instructions were to record consistently all completed calls they had made, including busy or wrong numbers and repeated calls. The time frame varied from one to three months.

The intent of the study was to inspect telephone usage patterns for a *rad* distribution. The collected data is surprisingly consistent. However, the limited number of subjects polled over a relatively short time period supplies enough data to support only general statements about usage patterns, and not specific quantitative measures.

Telephone usage by single users was surveyed and compiled, with the results reported in Table 5.1. The repetitive rate over all users is 51.6%, a figure which includes subjects who recorded very few calls. The weighted average repetitive rate — the repeat probability of any single call over all users — is slightly higher at 57%. This definitely places telephone number access within the *rad* domain, as it exceeds the rates found in *man* statistics. The range, which describes the difference between the users with the highest and lowest rate, covers almost 50% of the possible spectrum; a figure once again similar to *man's*. However, the given range result may be misleading, for the repetitive rate limits of active users — those who had made one hundred or more calls — are 50% to 72.7%, supplying a much tighter range of 22.7%.

Of interest is the relationship between repetitive rate and the number of phone calls dialed by a single user. The five users who had made over one hundred calls were analyzed and plotted in Figure 5.1. The graph indicates that the degree of repetition rises quickly over the first twenty calls and less quickly up to one hundred calls. The repetitive rate then seems to approach an equilibrium. Rises and declines are now slight. One interpretation of the graph is that users repeat phone numbers almost immediately, as shown by the rapid initial rise. Secondly, some calls are probably repeated over a slightly longer period of time, as revealed by the slow but steady increase in the middle of the curve. Finally, what is surprising is the apparent cessation of increase in repetitions after eighty calls, indicating that a high number of "once-only" calls exist, even after a lengthy time period. Clearly, there are three general categories of calls; highly popular numbers which are called quite often, moderately popular ones

Measure	Result
Average Repetitive Rate	51.6%
Weighted Average Rate	57.0%
Maximum Repetitive Rate	72.7%
Minimum Repetitive Rate	24.2%
Range (Max - Min)	48.5%
Standard Deviation	14.0
Average Number of Calls	85.8
Maximum Number of Calls	313
Minimum Number of Calls	23

Table 5.1: Telephone usage statistics

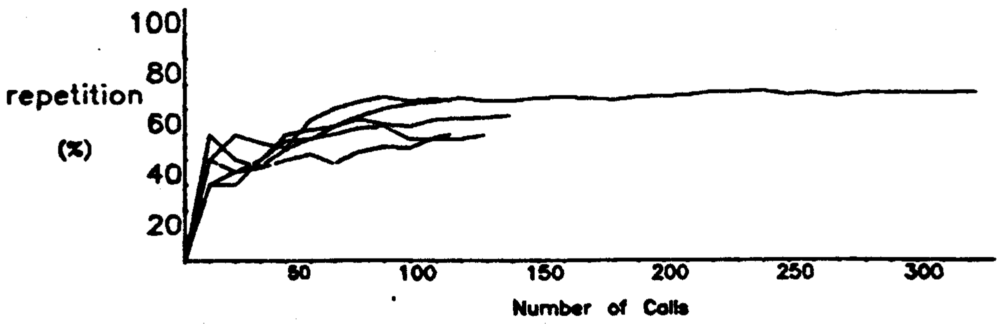

Figure 5.1: Relation between repetitive rate and number of phone calls

called infrequently, and once-only calls which are never or very rarely repeated.

A second analysis looked at the distribution of the repetitive rate over all calls for single users. Each subject's calls were ranked by frequency, and the three most active are illustrated in Figure 5.2 (also reported in Witten, Cleary and Greenberg, in press). The vertical axis shows the number of calls, normalized to one for the most popular number; while the horizontal axis shows the rank ordering of popularity. The continuous curve is a true Zipf distribution (Zipf, 1949), normalized in the same way. Of particular importance in the curves is the decreasing trend in the repetitive rate over the numbers, indicating the diverse spectrum between highly and rarely repeated numbers. This implies that an access method must give different degrees of preferential treatment over all numbers, with high frequency calls given easy access, perhaps at

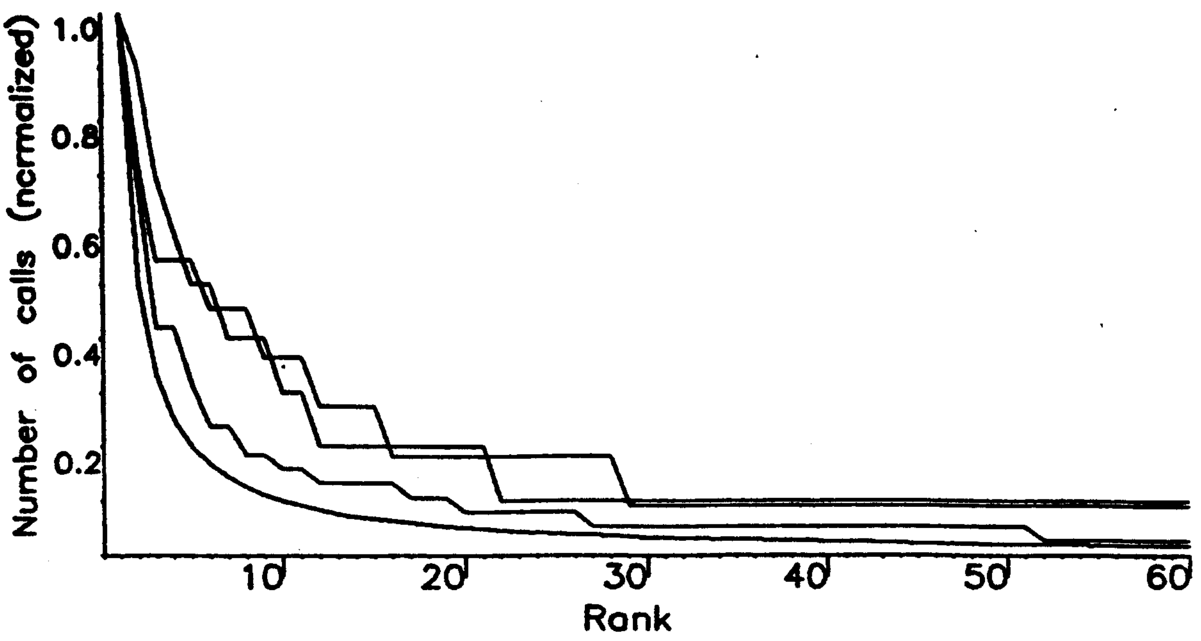

Figure 5.2: Sample distributions of personal telephone usage, compared with Zipf

the expense of low frequency ones. The same decreasing trend can be seen in the Zipf curve, although it is significantly more pronounced than in the telephone usage distribution. Nevertheless, the Zipf distribution provides a plausible model of the repetitive effects found in highly variable real-life usage patterns. Witten, Cleary and Greenberg (in press) examined a crude approximation to the entropy of this distribution for a large number of entries. They report that potential exists for frequency-based menu-splitting algorithms to reduce by half the average number of user selections required by a frequency-independent technique, if the directory is sufficiently large.

In summary, this modest study indicates that telephone usage exhibits even greater repetitive access than is found in *man*. A case is now made for incorporating the suggestions proposed in the previous chapter to a personalized telephone system, as discussed below.

5.2 The Personalizable Directory

This section examines a menu-driven selection process for retrieving entries from a large ordered *rad* list such as a telephone directory[1]. It pre-supposes that the user possesses a machine-readable directory, say that for the city in which he lives, which is distributed en masse and is in no way personalized. Menu-based schemes for retrieving items from the directory are devised and assessed. A full alphabetic keyboard and typing ability are not assumed.

[1]Much of this section appeared in a slightly different form in Witten, Greenberg and Cleary (1983), as presented at Graphics Interface '83, Edmonton.

information about accesses which have been made so far is automatically stored in a user profile list. This profile provides information allowing rapid access to popular entries by decreasing the number of menu pages which must be scanned.

The presentation system displays a *range* of names as each menu item. The first-level menu divides the name space into regions, the number of which is given by a pre-defined "menu size" parameter. One of these regions is selected by the user at the first level. The system then takes the indicated range and calculates a new, second-level menu by splitting it into sub-ranges. A further selection is made, and the procedure is repeated. Eventually, some of the menu items will be single names rather than regions, and when one of these is selected the search terminates. Figure 5.3 represents an unoptimized uniform menu tree hierarchy containing 20 name entries, produced by subdividing the name space as equally as possible at each stage with a menu size of 4. The number following each name shows how many menu pages were scanned before the final selection is reached. The corresponding first-level menu is shown in Table 5.2.

What is novel about the scheme is that the regions are selected at each stage not to cover an approximately equal range of names, but rather to divide a ***probability distribution***, defined on the name space, into approximately equal portions. The probability distribution is updated at the end of each selection process to reflect a "popularity" measure on the names. Although when the system is loaded in an unprimed state all names have equal popularity, the act of selection will increase the popularity of selected names. Thus the user will be directed more quickly to names which have already been selected — especially if they have been selected often and recently — than to those which have not. Indeed, depending on the menu size and the pattern of selections, it may be that names sought frequently appear directly on the first-level menu; and if not, they may almost certainly appear on the second-level one.

The price to be paid for this personalization is that novel names must obviously take longer to select than they would otherwise. However, the penalty can be made very small if suitable algorithms are used for probability updating. Figure 5.4 represents a menu hierarchy reflecting a "popularity" distribution. Highly popular names, such as Graham and Zlotky, appear immediately on the first level menu (Table 5.3). Other, less popular names are accessed on the second level menu, while the remainder are relegated to the third level. The menu hierarchies shown here are certainly not optimal. A discussion on frequency-based menu-splitting algorithms can be found in Witten, Cleary and Greenberg (in press).

5.2.1 An example of a personalized telephone directory

Imagine looking up a telephone number in a directory with N entries, using a menu with M items and a M-key keypad to select the menu item. Typical values are $N=250{,}000$, for a city with a population of 600,000; and $M=16$ for a 16-key pad.

We will consider a presentation system which displays a *range* of names as each menu item; an example of the first-level menu is shown in Figure 5.5. This divides the name space into M regions, one of which is identified by the first-level selection. The system will then take the indicated range, split it into M equal parts, and await a further selection. Eventually, some or all of the ranges will have been narrowed down into unique entries, and the final selection will indicate which name is sought. There are some obvious problems here. For example, the beginning and end of each menu range need not be presented on the screen, and doing so — as

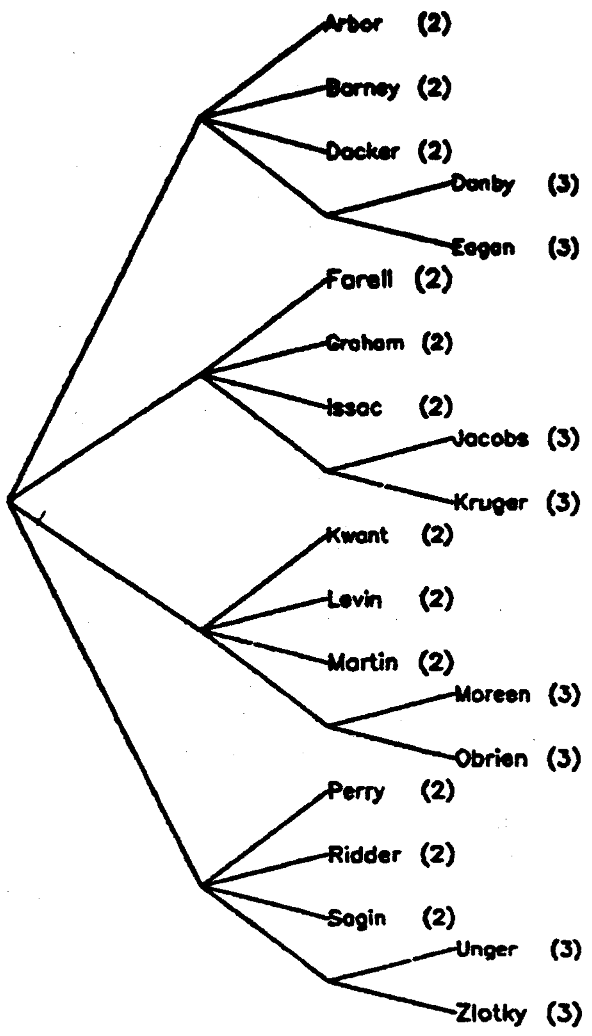

Figure 5.3: Menu tree generated by uniform subdivision

in the Figure — will likely delay the user for he has more information to scan. Furthermore, name ambiguities, when the address will have to be shown to allow disambiguation, will certainly occur. If several people have the same name and initial, the system may have to resort to a menu, or sequence of menus, showing all those people, because the address field may not be listed in any specific order in the directory. Although these problems affect the presentation and testing details, they will not affect the algorithm for menu construction, and are thus ignored for the rest of this discussion.

5.2.2 Uniform selection

With M menu items, each selection reduces the range of names to $1/M$'th of its former value. Hence with N names the process will terminate after

1	Arbor	—	Eagan
2	Farell	—	Kruger
3	Kwant	—	Obrien
4	Perry	—	Zlotky

Table 5.2 First-level menu for Figure 5.3

$$k_n = \frac{\log N}{\log M} \quad \text{selections,}$$

or 4.5 with our example values. That is, some retrievals will require 4 selections, while others take 5.

In over 50% of the cases, as estimated in Section 5.1, the act of selection provides information about items that are likely to be selected in the future. As far as retrieval is concerned, this information can be modeled in terms of a non-uniform distribution over the name space. (The discussion above assumed a uniform distribution where the probability of selecting any particular name is $1/N$). Instead of splitting the name space into M equal parts at each stage, it is the ***probability*** distribution that is split equally. This means, for example, that any name whose probability has grown to over $1/M$ will be present, in a range of its own, on the first-level menu.

5.2.3 Bimodal selection

As telephone usage displays a ***rad*** profile, the suggestions in Chapter 4, as summarized in Table 5.4, will be adopted and noted in the following discussion. Suppose you have F "friends", who have each been accessed the same number of times. The remaining entries, $N-F$, have never been retrieved. According to some algorithm for probability updating, which will be discussed later, suppose that the probability associated with each friend is p, and that of each non-friend is q, with

$$Fp + (N-F)q = 1.$$

Now it is not the name space but the ***probability range*** which is reduced to $1/M$ of its former value by each menu selection. Thus to access a friend will take k_F selections, where

$$k_F = -\frac{\log p}{\log M}.$$

It will be easy to calculate the number of selections to access a non-friend:

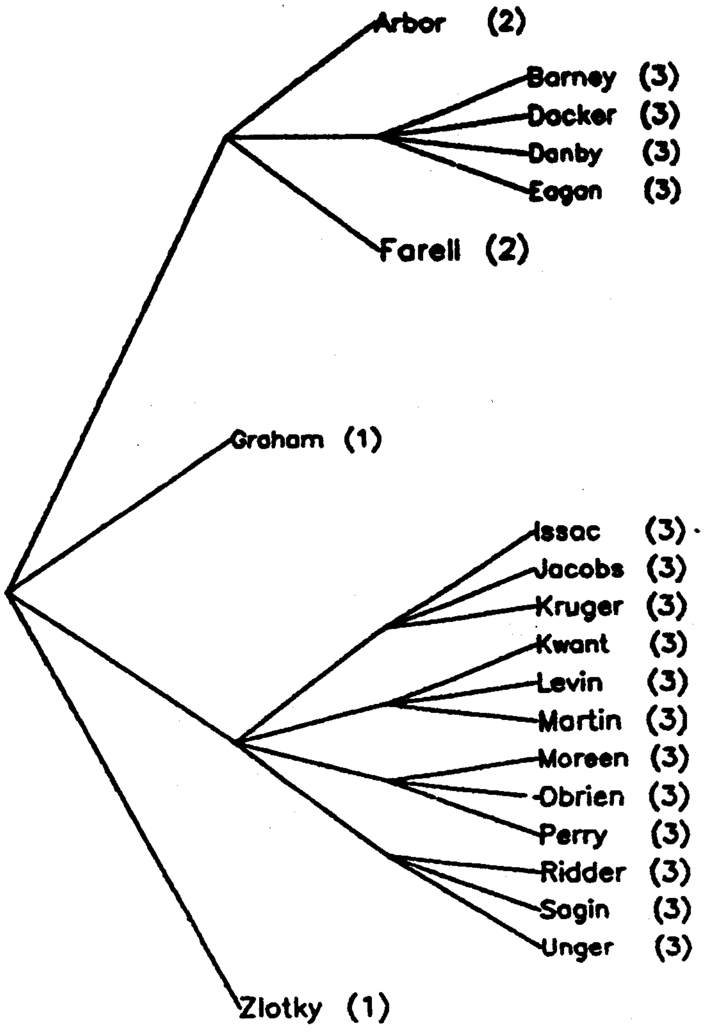

Figure 5.4: Menu tree reflecting popularity of items

$$k_n = \frac{\log N - \log(1 - FN^{-1}) - \log(1 - Fp)}{\log M} .$$

Under the reasonable assumption that $F \ll N$, and the somewhat more debatable one that $Fp \ll 1$, the Taylor expansion of log is used to reduce the expression to

$$k_n = \frac{\log N - FN^{-1} + Fp}{\log M} .$$

It can be argued that FP, the total probabilities assigned to friends, can actually take up a good proportion of the total distribution for a use geared heavily toward friend access. But even in this situation, the final equation still remains a good estimate of k_n. What about the value of p? Suppose that we wish to reduce the selection count for friends to half its value for a uniform distribution (Suggestion 1).

1	Arbor	—	Farell
2	Graham, John		
3	Issac	—	Unger
4	Zlotky, Ivan		

Table 5.3 First-level menu for Figure 5.4

1	A Abode Ltd	—	Austin Cal
2	Austin D	—	Bracal Mary
3	Bran W	—	Church of Pentecostal
4	Church of Prophecy	—	Dilling Ltd
5	Dillman A	—	Frankles Flowers
6	Frankiewicz S	—	Handfield E
7	Handfield RS	—	James R M
8	James R N	—	Lang J E
9	Lang J N	—	Martin J A
10	Martin J B	—	Mt Royal College
11	Mt Royal Day Care	—	Peters Earl
12	Peters Elwood	—	Roth Energy Ltd
13	Roth F	—	Sooner Pipe Ltd
14	Soong Jogh	—	True West Land Ltd
15	Truefitt C S	—	WestCan Graphics
16	WestCan Office Ltd	—	Zywoto A P

Figure 5.5. Example first-level menu, $M = 16$

$$k_F = \frac{1}{2} k_u .$$

Then it follows that $p = N^{-1/2}$. Plugging in numerical values ($N = 250000$, $M = 16$), with 100 friends, leads to a selection count of just over 2 for friends and only 3% above 4.5, the uniform distribution value, for non-friends (Suggestion 1 and 2).

1. Access to repeated items must be treated preferentially.
2. First time accesses to entries are not noticeably longer than those in a non-personalized system.
3. Personalization should be done at an individual level.
4. Personalization schemes should handle a wide variety of usage patterns.
5. The system should be primed to a sensible default state.
6. Items with different access frequencies should be graded according to that rate.

Table 5.4 Suggestions for personalization of *rads*.

It is important not to let a predominance of "friend" accesses force the selection count to grow to unacceptably large values for names which have not yet been retrieved (Suggestion 2). Suppose, for example, that at most $2k_u$ selections are to be tolerated. This is easily achieved by ensuring that all probabilities in the distribution exceed N^{-2} .

5.2.4 Menu-splitting algorithms

As another part of this research project, we have investigated menu-construction algorithms for accessing items from an ordered dictionary whose frequency profile of access is specified (Witten, Cleary and Greenberg, in press). The aim is to minimize the average number of selections. Despite the apparent simplicity of this problem, optimal menu construction is surprisingly difficult. For example, if the dictionary entries are accessed uniformly, theoretical analysis leads to a selection algorithm different from the obvious one of splitting ranges into approximately equal parts at each stage. The optimal menu tree can be found by searching, but this is computationally infeasible for any but the smallest problems.

Several algorithms for selecting items from an ordered dictionary with specified probabilities have been investigated. They all involve the selection of ranges of dictionary entries from a succession of menus, the range narrowing progressively until it contains just the desired item. They differ in their treatment of rounding in the menu-splitting process, and lead in general to quite different menu trees.

In the case of uniformly-distributed dictionary entries, which is amenable to theoretical treatment, the simplistic method of splitting each range into as many parts as there are menu items was found to be inferior — often grossly inferior — to a method which instead maximizes the utilization of leaves in the menu trees. For other distributions, theoretical analysis is intractable, although the performance of menu-splitting algorithms can be bounded. Three quite different algorithms have been investigated by computer simulation with a Zipf distribution access profile. This distribution approximates word usage statistics in natural language (Zipf, 1949); forming a good model of some artificial phenomena (Peachey et al, 1982), and an approximate model of certain natural phenomena, such as telephone usage. The performance of the three algorithms is remarkably similar, despite the fact that our limited experience with the optimal menu tree suggests that they leave some room for improvement. Until better methods are found, however, the simple process of iteratively splitting the probability range at each stage into regions whose probabilities are as similar as possible appears to be as good as any.

The actual implementation of the iterative menu splitting algorithm introduces the interesting side effect of popular names appearing at the splitting boundaries. In presentation terms, this implies that popular names frequently occur as a range delimiter in menus high in the hierarchy. This side effect is of benefit, for as a user is more likely to search for a popular name, it is probably easier for him to find the correct range if the name itself is a delimiter.

5.2.5 *Algorithm for updating probabilities*

Now we need to consider algorithms for updating the probabilities when an item is retrieved. The "friends"/"non-friends" bimodal situation was introduced above, in the context of retrieval. An algorithm has been designed for updating probabilities which

- ensures that no entries require more than a preset maximum number of selections (Suggestion 2);
- when a non-friend is accessed, making him a friend increases immediately the probability associated with him to take account of the fact that he is now much more likely to be accessed again (Suggestions 1 and 6);
- ensures that the probability associated with friends who are not accessed decays slowly (Suggestion 6);
- when a friend is accessed, increases the probability associated with him to re-enforce his popularity (Suggestion 6).

The easiest way to satisfy the first point is to decide in advance the maximum number of selections which can be tolerated for non-friend accesses. Suppose this is μk_u, where, as before, k_u is the selection count for a uniform distribution, and μ is a constant greater than one. Then non-friend probabilities are maintained at $N^{-\mu}$, ensuring that non-friend accesses take μk_u selections (on average). Of course, an exception must be made at the beginning when there are no friends; perhaps by setting probabilities to N^{-1} initially and changing them to $N^{-\mu}$ as soon as the first person is accessed (Suggestion 5).

If there are F friends this leaves the total probability weight assigned to friends as

$$1 - (N-F)(N^{-\mu}),$$

This ensures an average selection count for friends which is significantly less than k_u if F is small compared with N (Suggestion 1,6).

The second condition is easily satisfied, when a new friend is made, by giving him the average probability over all existing friends and decreasing their probabilities by the appropriate amount to ensure conservation of total probability.

The third and fourth conditions require some kind of adaptation algorithm on the probabilities, when existing friends are accessed. When a friend f is accessed, a procedure like

$$\begin{aligned} p_f &\leftarrow ap_f + 1-a \\ p_g &\leftarrow ap_g \quad \text{for friends } g \neq f \end{aligned}$$

appears to be suitable. This conserves probability if the total probability ***over friends alone*** is equal to one; which of course will not be the case due to the non-zero probability assigned to non-friends. A simple implementation maintains values for friends alone, which sum to one; and make an appropriate adjustment to the values before they are actually used as probabilities in the retrieval algorithm. Figure 5.6 shows details of computationally simple retrieval and update algorithms.

The parameter a is a measure of how quickly friendships fade away. a should be chosen slightly less than one; the closer it is to one the slower friendships will fade. The value of a should depend on the variety of usage patterns that exists between users (Suggestion 4). A more complex implementation could examine the individual's usage patterns and adjust a accordingly.

It is necessary to check when decrementing a friend's probability that it has not become less than that of the non-friends (Suggestions 2 and 4). Indeed, it may be desired to avoid gradual growth of the friend list over a long period of time. This can be done by removing friends whose probabilities have fallen below a certain value (say N^{-1}), and making appropriate adjustments to all other friends' probabilities.

5.2.6 Summary and the next step

We have demonstrated how automatic user modeling in a sequential directory can reduce the average number of selections needed to retrieve entries. The method described has the important advantage of being applicable to displays of any size, including those whose size changes within the retrieval process. This is useful in a dynamic window-based computer interface. There is room for research into practical selection algorithms which achieve the theoretical, entropy-based, minimum number of selections without excessive searching (Witten, Cleary and Greenberg, in press). However, simple methods come quite close to realizing the full potential of the method for reducing selections.

The selection method requires an estimate to be available of the frequency profile with which items are accessed. Such estimates are complicated by the fact that the distribution changes with time, and that the information available is somewhat sparse in relation to the large number of possible directory accesses. We have given an example of a plausible estimation technique for a bimodal distribution of friends and non-friends which conform to suggested guidelines for personalizing ***rad***'s.

This simple personalizable directory scheme raises a number of difficult issues in automatic user modeling. These include the user's perception of a dynamic system, the illusion of user control, the measurement of user satisfaction, and many others as discussed in Chapter 3. The directory scheme can now be used as a test-bed to examine such issues in the context of an adapting system through simple pilot experiments. Through these studies we hope to gain some insight into the human factors implications of automatic user modeling, and to develop

---

Keep a list of pointers to friends f who have already been accessed,
together with a weight $w(f)$ for each in the range $[0,1]$.
The update algorithm will ensure that the weights sum to 1 over all friends.

Retrieval.

```
Use the following probabilities:
        if $F=0$, probability for each name is $\frac{1}{N}$
        otherwise,
            if friend, probability is $\frac{1}{N^\mu} + w(f)(1 - \frac{N}{N^\mu})$
            else probability is $\frac{1}{N^\mu}$ .
```

Update.

```
If a new friend has been made, put him on the friend list
and update $F$. Then
        initialize his weight to $\frac{1}{F}$
        decrease all other weights by $\frac{1}{F(F-1)}$.
If an old friend $f$ has been accessed again,
        $w(f) \leftarrow aw(f) + 1-a$
        $w(g) \leftarrow aw(g)$ for other friends $g \neq f$ .
In either case, check if any $w(f) \leq \frac{1}{N}$ ; if so
        delete $f$ from the friend list, decrementing $F$
        increase weights of other friends $g$ by $\frac{1}{F} w(f)$.
```

Figure 5.6 Retrieval and update algorithms

meaningful guidelines to supplant previous wholly intuitive design methodologies.

5.3 Comparison of Personalized and Non-Personalized Directory Systems

We now have some tools suitable as a basis for investigating the effectiveness of personalization in a directory system through human factor experiments. The menu-based schemes underlying the bimodal selection process allow presentation of two systems differing only in their inclusion of personalization, simply by choosing to maintain or ignore the user profile. This allows comparison between the personalized directory, which uses dynamic menus, and the static menu structure of the non-personalized control.

Another fundamental tool available is data collected on actual telephone usage (summarized in Section 5.1). A data segment suitable for incorporation in the experiment has been extracted from the compiled findings. This segment will be used to prime the personalized system to the equilibrium state described in Section 5.1.2, and will be followed by selections for the subject to access.

However, some points must be clarified before the comparison of the two systems can be performed. The presentation details are an important concern for human factors evaluation. A display which is difficult to scan rapidly will give the user a long selection time per menu. If the automatic system is then measured against a static control which has a fixed and memorizable pathway, the scanning delay will weigh against the personalized system. The Appendix reports a human factors investigation of six possible menu displays. It offers clear guidance as to which menu format provides the best human performance in terms of scanning speed and error rate. The upper range, non-truncated delimiter menu — as described in the Appendix — is selected for the current study.

The preliminary study described in Appendix One also investigated performance of computer novices and experts in the menu search task. Novices were found to have slower scanning speeds and greater sensitivity to menu displays than experts. In addition, more variability is present with the novice group as compared to experts. Thus the current study selected computer experts as subjects in an effort to minimize the subject variable. The subjects are defined as ***foreign users***; those who are novices with respect to a given system but expert with computers in general.

Finally, differences between subject performance in scanning root menus and menus buried deep in the tree are reported in Appendix One. The results suggest that the traversal of alphabetic menu hierarchies should avoid, as much as possible, descending deep into the tree, for subject efficiency deteriorates with depth. This result favors the personalized directory structure, which minimizes descent into the menu hierarchy.

The comparison of the personalized directory system against a static control is now described. It tests for the following null hypothesis:

- Personalization of the telephone directory system does not affect selection speed and error rate per individual menu and per trial, where a trial is defined as the complete process of locating the final menu leaf after being initially supplied with a name.
- The order of system presentation to a subject has no effect on his performance.

The ***method***, which describes the design and physical makeup of the experiment, is discussed first. This is followed by the ***results***, a strictly statistical evaluation of the generated data. Finally, the ***discussion*** and ***conclusions*** interpret the results with respect to the specific and general context of the personalizable directory.

5.3.1 Method

Subjects. The subjects are twenty-six paid volunteers (university students). All were solicited from senior computer science courses. Since none had previous experience with the personalizable directory system, they are labeled as "foreign users". Some subjects of this study had previous exposure to the menu system: they were randomly mixed with non-experienced subjects.

Subject Use. This experiment is a 2-level (adaptation type) by 2-level (order) mixed factorial design (Figure 5.7). Each subject is assigned to both levels of adaptation type and only one order level, giving a total of 13 subjects per cell.

Apparatus. A Corvus Concept microcomputer connected to a VAX-11/780 is used to display all material for the experiment on a high resolution bit-mapped screen. Keystrokes are timed locally on the Corvus so that precise measurements can be made (to within 10 msec). Instructions to subjects are given first verbally and then on-line. The data base used was derived from the University of Calgary telephone directory and contained 2611 unique entries.

Data Selection. Data used for the experiment constitutes a profile identical to the first 160 calls made by one subject in the sample study (Section 5.1.2). This profile is randomly mapped on to the directory data base. The repetitive rate stabilizes to approximately 63% after ninety calls. The system is primed with the first 122 profile entries. The following 38 calls are presented to the subjects. Use of the data in the prescribed manner gave the following system characteristics: subjects are forced to go through 4 menu selections for each name in the static system, and an average of 2.7 selections for the dynamic system.

Design. Subjects are randomly assigned to one of the two order groups. Each subject of each group is exposed to two sets of menu systems employing different adaptive methods.

Procedure. Each subject is exposed to two sets of menu systems, one which uses a dynamic adaptation algorithm and one which does not. Each set presents instructions on how to use that particular system, followed by a practice session of eight trials and a test session of thirty trials. Each trial is composed of five parts:

		Order	
		Dynamic First	Static First
Adaptation Type	Dynamic	S1−13	S14−26
	Static	S1−13	S14−26

Figure 5.7: Mixed Factor Anova Design

- A name is presented to the subject, who reads it and presses <return>. Timing of the trial begins.
- A menu display appears below the name and is scanned by the subject. His task is to identify the range or the unique entry which contains the supplied name.
- If the name is not a unique entry on the menu, the subject will select the menu item which indicates the correct range. The scanning time of the menu is noted. The previous step is then repeated with a new menu display representing a branch down the menu tree.
- If the name is a unique entry, the subject will select it, successfully ending the trial. Timing of the trial ends.
- The system will indicate errors to the subject, and ask that he makes another selection. Errors and their time of occurrence are marked.

The menus are the same in both sets and across all subjects.

Measures. The dependent measures are the scanning time per trial, error rate per trial, scanning time per menu and error rate per menu. The independent measures are adaptation type and presentation order. Scanning time per trial is the time in seconds between the appearance of the first menu and the successful location of the unique entry in the final menu. Error rate per trial is the percentage of selection errors made in a given trial. Scanning time per menu is the time in seconds between the appearance of a menu and the successful location of the name in that menu. Error rate per menu is the percentage of selection errors made in a given menu. In addition, a questionnaire asking subjects which system they preferred is given for comparison with the quantitative result.

Motivation. It is desirable to keep the subject's motivation level constant to avoid uncontrolled variation within a subject. A cash prize is therefore awarded to the subject with the best performance, where performance is a combination of error rate and scanning time.

5.3.2 Results

Scanning speed and error rate were analyzed independently through the use of the analysis of variance statistical package (P4V) supplied by BMDP statistical software (Dixon *et al*, 1981). Table 5.5 provides an anova summary table of all results. All starred F-ratios in the table indicate a statistically significant result at or beyond the 0.05 level. For those not familiar with the anova concepts and terms, a detailed explanation of the subject can be found in Kirk (1968).

Scanning Speed and Error Rate per Trial. A 2X2 analysis of variance (Kirk, 1968) on scanning speed per trial revealed a significant main effect for adaptation type and an interaction between order and adaptation type (Table 5.5).

Post-hoc comparisons among cells of the adaptation type X order interaction matrix, using tests of simple main effects (Kirk, 1968), showed significant differences for adaptation at both levels of order, whereas order was only significant with the static adaptation type. Figure 5.8 illustrates the magnitude of the adaptation type/order interaction. There is a relatively small effect of order, from zero to two seconds, contrasted with the large effect of adaptation type, from five to seven seconds. The nature of the interaction allows us to consider, for all practical purposes, the main effect of adaptation independent of the interaction. The cell means of significant main effects on scanning speed are summarized in Table 5.6.

Dependent Variable	Source	Degrees Freedom	Sums of Squares	Mean Squares	F-ratio
Scanning Speed Per Trial:	*Between:*				
	Order (O)	1, 24	10.629	10.629	1.81
	Subj. w.groups		141.055	5.877	
	Within:				
	Adaptation Type (A)	1, 24	419.809	419.809	452.91**
	A X O	1, 24	14.217	14.217	15.34**
	A X subj w.groups		22.246	0.927	
Error Rate Per Trial:	*Between*				
	Order (O)	1, 24	167.545	167.545	0.93
	Subj w.groups		4324.110	180.171	
	Within:				
	Adaptation Type (A)	1, 24	2777.220	2777.220	35.62**
	A X O	1, 24	85.504	85.504	1.10
	A X subj w.groups		1871.214	77.967	
Scanning Speed Per Menu:	*Between:*				
	Order (O)	1, 24	0.616	0.616	1.20
	Subj w.groups		12.311	0.513	
	Within:				
	Adaptation Type (A)	1, 24	0.389	0.389	7.26**
	A X O	1, 24	0.942	0.942	17.58**
	A X subj w.groups		1.287	0.536	
Error Rate Per Menu:	*Between:*				
	Order (O)	1, 24	11.962	11.962	0.77
	Subj w.groups		373.356	15.556	
	Within:				
	Adaptation Type (A)	1, 24	76.182	76.182	12.51**
	A X O	1, 24	4.373	4.373	0.72
	A X subj w.groups		146.184	6.091	

* $p < .05$
**$p < .01$

Table 5.5: Anova Summary Table For All Dependent Variables

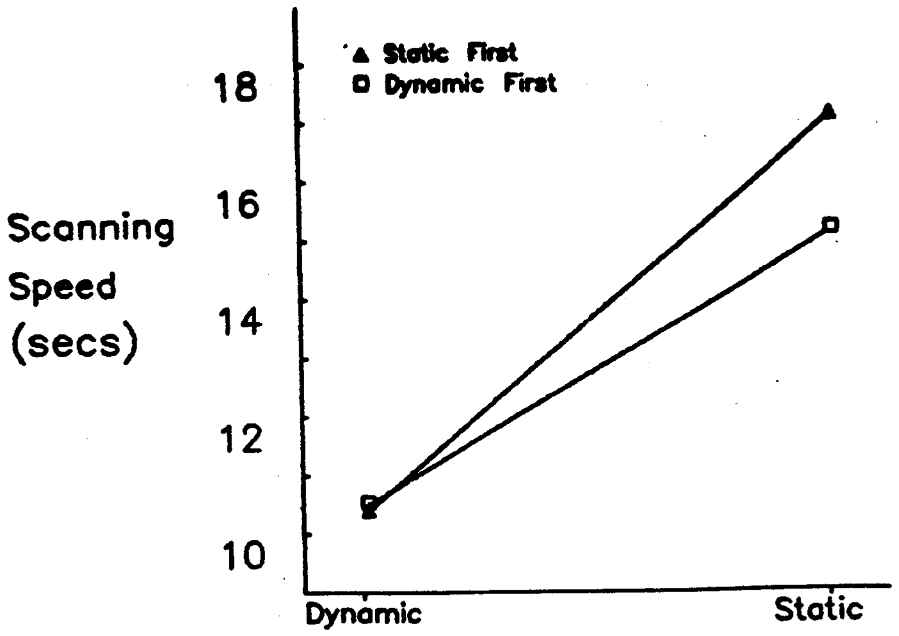

Figure 5.8: Order/Adaptation Interaction (Trials): Scanning Speed vs. Adaptation Type

Similarly, a 2×2 analysis of variance (Kirk, 1968) performed on error rate per trial revealed a significant main effect for adaptation type (Table 5.5). The cell means of significant main effects on error rate are summarized in Table 5.6.

Scanning Speed and Error Rate per Menu. A 2×2 analysis of variance (Kirk, 1968) on scanning speed revealed a significant main effect for adaptation type and an interaction between order and adaptation type (Table 5-5).

Post-hoc comparisons among cells of the adaptation type × order interaction matrix, using tests of simple main effects (Kirk, 1968), showed significant differences for adaptation at the static first level of order, whereas order was only significant with the static adaptation type. Figure 5.9 illustrates the magnitude of the adaptation type/order interaction. The nature of the interaction precludes the analysis of any significant main effects.

Similarly, a 2×2 analysis of variance (Kirk, 1968) performed on error rate per menu revealed a significant main effect for adaptation type (Table 5.5). The cell means of significant main effects on error rate are summarized in Table 5.6.

Factor	Level	Type	Scanning Speed (Seconds)	Error Rate (%)
Adaptation	Dynamic	Trials	10.37	10.26
Adaptation	Static	Trials	16.06	24.87
Overall		Trials	13.22	17.56
Adaptation	Dynamic	Menus	—	3.79
Adaptation	Static	Menus	—	6.31
Overall		Menus	3.93	5.01

Table 5.6: Cell means of significant main effects and their levels

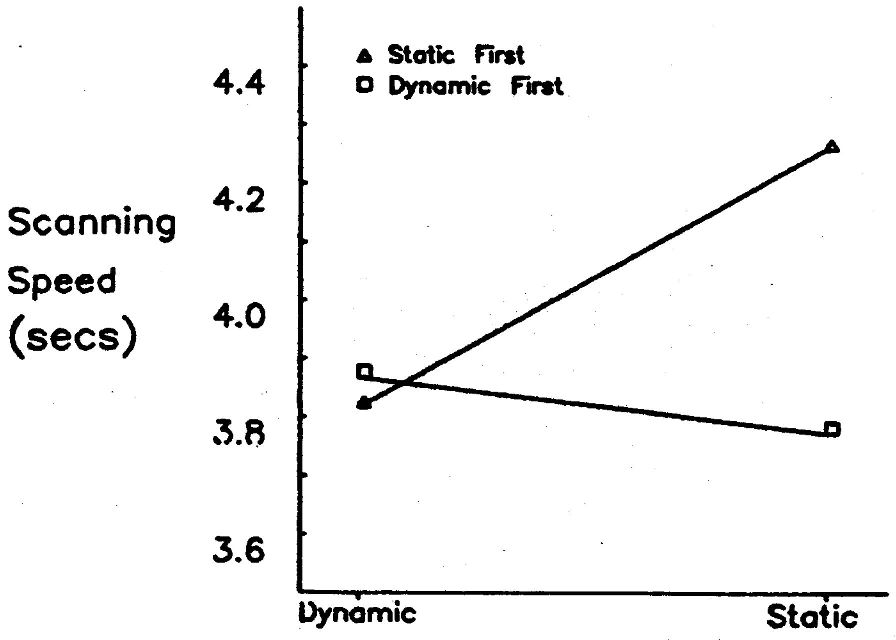

Figure 5.9: Order/Adaptation Interaction (Menus): Scanning Speed vs. Adaptation Type

After task completion, subjects were queried about their system preference. Eighteen subjects reported a strong preference for the dynamic system, three were undecided and five weakly preferred the static system.

5.3.3 Discussion.

This study compares two systems which differ in their ability to adapt to user input. Two levels of dependent variable are examined: a task (trial) level which is the successful location of a unique name through a chain of menus, and a menu level which is the interim search through any single menu. Evidence supporting rejection of the stated null hypothesis is as follows:

1. The dynamic directory has a significantly faster trial completion time than the static directory (Table 5.6).
2. The dynamic directory has significantly fewer errors per trial than the static directory (Table 5.6).
3. Trial completion time is lessened in the static directory when the subject has had the experience of using the dynamic directory. However, the magnitude of improvement is much less than that shown by use of the dynamic system (Figure 5.8).
4. Menus generated via the dynamic directory are initially faster to scan than static menus. However, no difference exists when the subject had previously used the alternate system (Figure 5.9).
5. Menus generated via the dynamic directory have significantly fewer subject errors than those generated via the static directory (Table 5.6).
6. Most subjects preferred to use the dynamic directory system. The subject comments indicate that this is due to the apparently shorter search paths for repetitive names, the lack of necessity to memorize, and the frequent appearance of the given name as a name delimiter in some of the high-level dynamic menus, even though further descent into the tree was still necessary.

5.3.4 Conclusion

The results of this experiment indicate that the personalizable directory is far superior to the non-personalizable directory, at both the trial and menu levels. Table 5.7 summarizes the degree of improvement of the personalized system over the static one. The task of locating a given person is shortened by almost six seconds (a 35% improvement) while the error rate is decreased from 25% to 10% (a 60% improvement). However, the smaller reduction in the actual number of menus scanned — from 4.0 to 2.7 (a 32% reduction) — indicates that factors other than individual menu performance are responsible for the superiority of the personalized directory.

At the menu level, fewer errors are made in the dynamic system than in the static one, with a reduction from 6.3% to 3.8% per menu (a 40% improvement). There are two possible reasons for this reduction. First, a subject's attempt to memorize pathnames in a static system may lead to increased errors. Secondly, as reported in Appendix One, subject performance degrades with his depth in the menu hierarchy, a condition which is minimized in the personalized system.

Task	% Reduction (static to dynamic)
Scanning time/trial	35%
Errors /trial	60%
Scanning time/menu	0%
Errors/menu	40%
Menus searched/trial	32%

Table 5.7: Degree of improvement (static to dynamic)

Subjective preference in choice of systems was heavily weighted toward a dynamic system, even though the subjects were never told that the system enabled them, on average, to traverse fewer individual menus. Thus, both the quantitative and qualitative results support the advancement of a personalizable directory system.

The university directory data base used in this experiment was small when compared to city directories with millions of entries. The personalized system can provide an even better theoretical performance in these very large directories. As noted in Section 5.1.2, this can approach a 50% reduction in the average number of menus scanned for a Zipf distribution over a large directory, as compared with 32% in the tested system. Thus the estimation of performance improvement noted here is probably conservative when compared to real life.

One counter-point should be explored. The subjects ran data simulating about ten days of telephone usage. A higher volume of data representing a longer time period would presumably have facilitated memorization of the relatively few highly popular entries, thereby reducing times in the static system. In this case, the key value of personalization would rest in improving access to the higher volume of moderately-accessed phone numbers which are not memorized. One can envision a real-life system allowing a user to ask for a menu of the nine most popular entries, deferring to the personalized directory system for all others.

6. Summary

Everything's got a moral, if you can only find it.
(Alice's Adventures in Wonderland - Lewis Carroll)

The main objective of this thesis, described in Chapter One, was "to determine the viability of an adaptive interface in an interactive computer system, thus providing a concrete refutation of objections raised by certain authors". To fulfill this objective, user modeling was defined and categorized, relevant issues were presented, and an application area was proposed and constructed as a test bed for examination of some of these issues.

Chapter One provided a framework for characterizing adaptive interfaces, first through the presentation of two possible system architectures, and then through defining a taxonomy classifying modeling into three types: canonical, explicit and automatic. Specific user problems in each domain were examined in Chapter Two. It is clear from the examples presented that all the noted personalized systems introduced further problems. Better systems merely had a better balance of trade-offs.

Chapter Three examined the literature on some of the fundamental issues in user modeling, and highlighted possible causes behind specific system problems. The controversy between authors defending their views prohibited the promotion of any particular stance, for most opinions were based on folklore, intuition and personal experience. The views presented are not necessarily wrong; rather, it is just the lack of empirical studies which minimizes their credibility. The most fundamental issue — whether or not adaptive user modeling is a viable alternative to static systems — remained unanswered.

The remainder of the thesis strove to resolve this issue. The approach was to explore, construct and test a simple system in which automatic personalization would give demonstrable benefits over a non-personalized approach. Success of the modeled system could support the viability of adaptive modeling in principle, whereas its failure would provide a basis for the empirical study of fundamental issues. To this end, the characteristics of repetitively accessed databases — an information system ripe for personalization — were defined and studied in Chapter Four.

Chapter Five presented the personalized directory, a menu-driven *rad* system with a simple automatic user modeling mechanism. Human factors experiments, based on characteristics of telephone usage as applied to the directory system, show that not only are significant benefits supplied through personalization, but subjects prefer using the adaptive system. More specifically, the average time taken to traverse a menu tree is reduced in the personalized directory — subjects are able to retrieve an entry faster. In addition, the average number of errors is smaller during both tree traversal and individual menu scans. When one considers the high penalty of errors in real life applications — users going down incorrect branches of the tree not only lose time but also suffer the risk of getting "lost" in the hierarchy — then the significance of reducing errors becomes apparent.

The Appendix presented a study of presentation details in menu displays. This study is important in its own right, for not only is the best menu format clearly supported by the results, but general effects of other factors, such as user experience and depth in a menu tree, are identified. Three techniques of delimiting ranges are contrasted. The ***upper range delimiter*** method — which displays only the alphabetically greater limit of a range — is superior on all counts, for subjects scan these menus faster and with fewer errors than any other menu style. Elimination of letter redundancy through truncation of delimiters has no effect at all; its use is left to the discretion of the designer. In general, novice users have slower scanning speed and greater sensitivity to menu display types than experts. But perhaps most importantly, user efficiency deteriorates with the depth of the menu hierarchy, a result favoring a personalized directory structure which minimizes descent into the tree.

This thesis supports, through empirical studies, the viability of user modeling in interactive computer systems. Many of the arguments against personalization, as presented in Section 3.2, have been disputed. It has been shown that concurrent modeling in a man/machine system need not destroy the success of the interface. In addition, an adaptive system has been successfully implemented and evaluated. It has been demonstrated that inaccuracies of model construction do not necessarily constrain the interface, for repetitions within the dialogue allow reinforcement of important model aspects at the expense of the rarely-repeated or erroneous aspects. Finally, it has been shown indirectly that dialogue determinancy is not necessarily compromised through personalization, even though automatic modeling assumes full system control of some dialogue aspect.

It would be worthwhile in the future to study adaptive systems in other menu-driven contexts. One direct application is a personalized address list displayed on an alphabetically-ordered simulated Rolodex-like device (Shneiderman, 1983). A pointing device positioned over a direction and speed bar would give users rapid control over "card" motion through the Rolodex catalog, while pop-up menus allow selection of the particular catalogue to be viewed (Figure 6.1). Many configurations are possible. The first presents explicitly-constructed Rolodexes, where the user creates a series of card files containing user-defined entries. The Figure illustrates a pop-up menu with four Rolodexes to chose from, one being the default directory and the other three — business, buddies and girlfriends — created by the user. A second Rolodex configuration contains only those entries judged as "popular" by the personalized directory algorithm presented in Chapter 5. Yet another is a combination of the first two; names noted explicitly by the user and automatically by the system are compiled into one card catalogue. Finally, the complete Rolodex directory can be automatically and explicitly personalized. When the user scans the catalogue, the system would pause on each entry for a time proportional to its "popularity", where explicitly-noted items are given relatively long pauses, automatically-noted entries are given ranked pauses, and all others are flashed by quickly. Direct manipulation of a Rolodex card file is likely to be a far superior interface than the personalized menu structure of Chapter 5, even though the underlying algorithms are identical. It would be interesting to redo the human factors experiment described in Section 5.3 contrasting the personalized and non-personalized Rolodex.

But it is the taxonomic menu, rather than the ordered menu, which is more prevalent in interactive computer systems. Another possible study could apply personalization to taxonomic menus, with the objective of minimizing the problems listed in Section 2.5. But automatic personalization of these menu types cannot be solved with the algorithm advanced by Section 5.2: another approach is needed. Assuming that taxonomic menu hierarchies display ***rad*** characteristics, it is possible to create dynamic menu structures which reflect only the portions

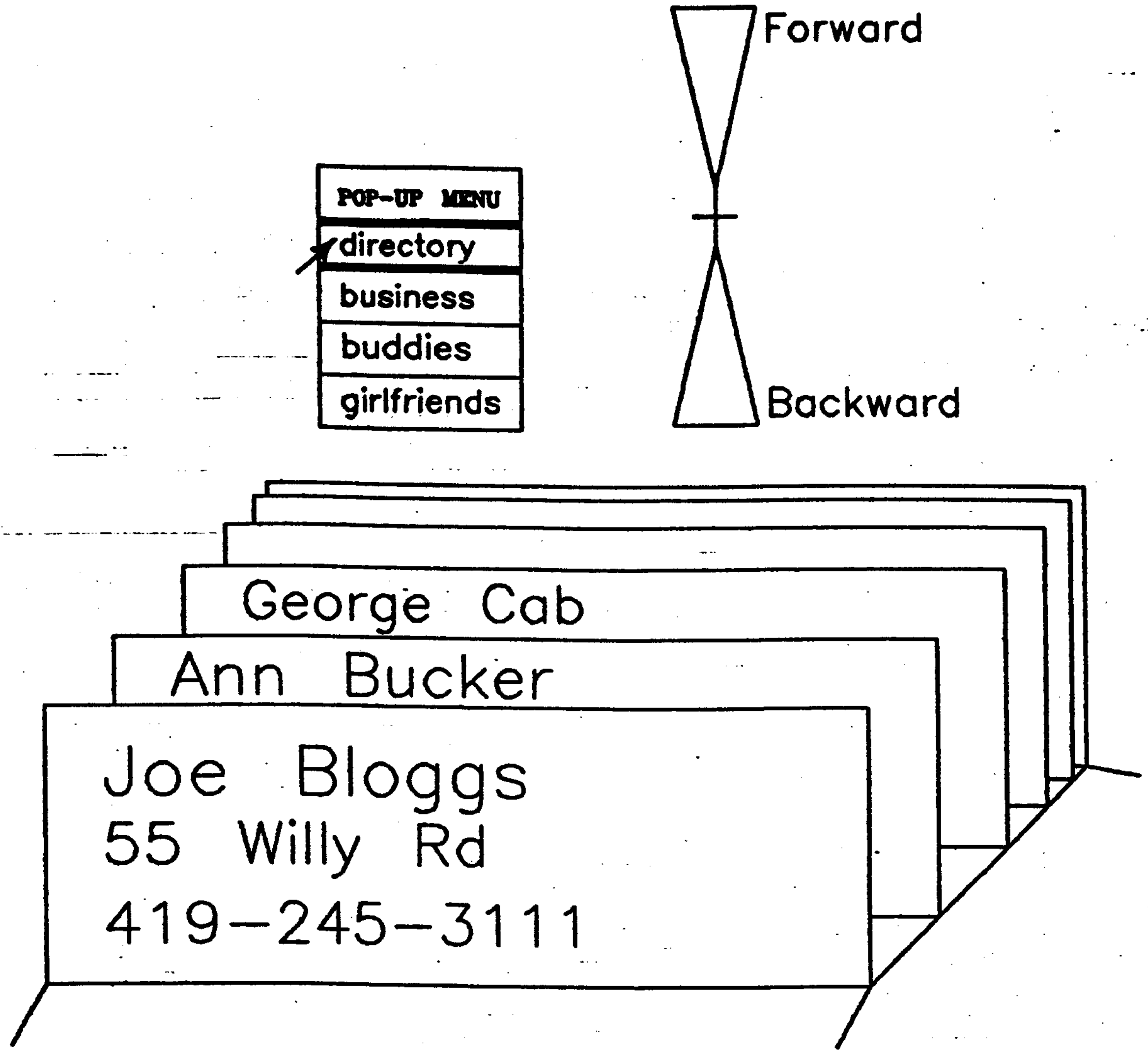

Figure 6.1: The personalized Rolodex directory

of the tree traversed previously. For example, a Videotext user interested in hockey but no other sports may traverse repeatedly a tree branch, such as: "sports-winter-televised-hockey-scores". The personalized structure could retrieve the scores as soon as sports was selected. If more than one leaf was previously accessed from a particular node, a dynamic menu would offer only the final leaves as selections, bypassing all intermediate nodes. But a more elegant approach uses keywords to access menu leaves randomly. A personalized system could build a keyword index by scanning all menu items traversed by the user. Direct access to a page is given if the user supplies a keyword appearing in that item. Referring to the previous example, the keyword "score" would display the leaf page of hockey scores, if the user had traversed that path at least once before. Multiple hits would provide a dynamic menu, ranked in "popularity" order.

There is no need to limit the work to *rads* or menus. We now know that the computer can adapt, albeit in a highly constrained manner, to every user. The two examples described above illustrate that personalized prototypes can be constructed. But we know little of the precise nature of human reaction to adaptive systems; further empirical studies must be performed. It is the results of these studies which will allow the designer/user team to create comprehensive adaptive dialogues of many types for users of interactive systems.

Bibliography

Abell, R. (1981) "Implementation of a Telidon system using UNIX file structures" in *The Telidon Book*, edited by Godfrey and Chang. Press Percopic Ltd., Victoria.

Bradley,J. (1982) *File and data base techniques*. CBS College Publishing, New York, USA.

Bramwell, B. (1983) "An automatic manual" MSc Thesis, Department of Computer Science, University of Calgary.

Cuff, R.N. (1980) "On casual users" *Int J Man-Machine Studies, 12*, 163-187.

Date, C.J. (1977) *An introduction to database systems*. Addison-Wesley Publishing.

Davis, R. (1979) "Interactive transfer of expertise: acquisition of new inference rules" *Artificial Intelligence, 12* (2) 121-157.

Dixon, W.J., Brown, M.B., Engelman, L., Frane J.W., Hill, M.A., Jennrich, R.I., and Toporek, J.D. (1981) *BMDP statistical software*. University of California Press, Berkeley.

Dumais, S. and Landauer, T. (1982) "Psychological investigations of natural terminology for command & query languages" in *Directions in human/computer interactions*, edited by Badre and Shneiderman, pp 95-110. Ablex Publishing Co., Norwood, New Jersey.

Eason, K.D. and Damodaran, L (1979) "Design procedures for user involvement and user support" *Infotech - Man Computer Communications*, London.

Edmonds, E. (1982) "The man-computer interface: a note on concepts and design" *Int J Man-Machine Studies, 16* (3) 231-236, April.

Ehrenreich, S.L. (1981) "Query languages: Design recommendations derived from the human factors literature" *Human factors, 23* (6) 709-725.

Engel F.L., Andriessen J.J., and Schmitz, H.J.R. (1983) "What, where and whence: means for improving electronic data access" *Int J Man-Machine Studies, 18*, 145-160.

Fitter, M. (1979) "Toward more natural interactive systems" *Int J Man-Machine Studies, 11*, 339-350 .

Foley, J. and Wallace, V. (1974) "The art of natural graphic man-machine conversation." *Proceedings of the IEEE, 62* (4) 462-471.

Fortin, J.V. (1981) "Telidon: A standard, a system, a network" *Text for the ASTED/CLA conference*, November.

Gaines, B. (1981) "The technology of interaction — dialogue programming rules" *Int J Man-Machine Studies, 14* (1), January.

Gaines, B. and Shaw, M. L. (1983) "Dialog Engineering" in *Computers and People series*. Academic Press, London, in press.

Gosling, James (1981) *Unix Emacs*. Carnegie-Mellon University, January.

Hagelbarger, D.W. and Thompson, R.A. (1983) "Experiments in teleterminal design" *IEEE Spectrum*, 40-45, October.

Hansen, W.J. (1971) "User engineering principles for interactive systems." *Proceedings of the Fall Joint Computer Conference*,, AFIPS Press, New Jersey.

Innocent, P.R. (1982) "Towards self-adaptive interface systems" *Int J Man-Machine Studies, 16* (3) 287-299, April.

James, E.B. (1980) "The user interface" *The Computer Journal, 23* (1), February.

Joy, William. (1980) "An introduction to the C shell" in *Unix Programmer's Manual, Seventh Edition, Volume 2c*. University of California, Berkely, California, November.

Kennedy, T.C.S. (1975) "Some behavioural factors affecting the training of naive users of an interactive computer system" *Int J Man-Machine Studies, 7* (6) 817-834, November.

Kiger, J. (1984) "The depth/breadth trade-off in the design of menu-driven user interfaces" *Int J Man-Machine Studies, 20*, In press.

Kirk, R. (1968) *Experimental design: Procedures for the behavioural sciences*. Wadsworth Publishing Company, Belmont, California.

Landauer, T.K., Galotti, K.M., and Hartwell, S. (1983) "Natural command names and initial learning: A study of text-editing terms" *Communications of the ACM, 26* (7), July.

Larson, J.A. (1982) *End user facilities in the 1980's*. IEEE Computer Society Press, Los Almitos, California.

Latremouille, S. and Lee, E. (1981) "The design of videotex tree indexes: the use of descriptors and the enhancement of single index pages" *Telidon Behavioural Research, 2*, Department of Communications, May.

Maguire, M. (1982) "An evaluation of published recommendations on the design of man-computer dialogues" *Int J Man-Machine Studies, 16* (3) 237-261, April.

Martin, T. (1980) "Information retrieval" in *Human interaction with computers*, edited by Smith and Green, pp 161-175. Academic Press, London.

Merriam, C and Webster, D (1964) *The new Merriam-Webster pocket dictionary*. Pocket Books of Canada, Ltd, Richmond Hill, Ontario.

Miller, G.A (1956) "The magical number seven plus or minus two: some limits on our capacity for processing information" *Psychology Revue*, *56*, 81-97.

Peachey, J.B., Bunt, R.B., and Colbourn, C.J. (1982) "Bradford-Zipf phenomena in computer systems"" *Proc Canadian Information Processing Society National Conf*, 155-161, Saskatoon, Saskatchewan, May.

Pew, R.W. and Rollins, A.M. (1975) "Dialog specification procedure" Report No. 3129, revised edition, Cambridge, Mass..

Raitzer, G.A., Vanderheiden, G.C., and Holt, C.S. (1976) "Interfacing computers for the physically handicapped – a review of international approaches" *Proc AFIPS National Computer Conference*, 209-216.

Rich, E. (1983) "Users are individuals: individualizing user models" *Int J Man-Machine Studies*, *18*.

Roberts, T.L. and Moran, T.P. (1982) "Evaluation of Text Editors" *Proceedings of human factors in computer systems*, Gaithersburg, Maryland, March 15-17.

Robertson, G., McCracken, D., and Newell, A. (1981) "The ZOG approach to man-machine communication" *Int J Man-Machine Studies*, *14*, 461-488.

Shaw, M.L.G. (1980) "On becoming a personal scientist: interactive computer elicitation of personal models of the world" in *Computers and People Series*. Academic Press, London, in press.

Shneiderman, B. (1979) "Human factors experiments in designing interactive systems" *Computer*, 9-19, December.

Shneiderman, B. (1983) "Direct manipulation: A step beyond programming languages" *Computer*, *16* (8) 57-69, August.

Stallman, R.M. (1981) "EMACS the extensible, customizable self-documenting display editor" *ACM Sigplan Notices – Proceedings of the ACM Sigplan SIGOA symposium on text manipulation*, *16* (6) 147-155, Portland, Oregon, June 8-10.

Thimbleby, H. (1980) "Dialogue determination" *Int J Man-Machine Studies*, *13*.

Tombaugh, J.W. and McEwen, S. (1982) "Comparison of two information retrieval methods on Videotex: Tree structure versus alphabetic directory" *Proceedings of human factors in computer systems*, Gaithersburg, Maryland, March 15-17.

Tompa, F.W. (1982) "Retrieving data through Telidon" *CIPS Session 82 conference*, Saskatoon, April.

Whalen, T. and Latremouille, S. (1981) "The effectiveness of a tree-structured index when the existence of information is uncertain" *Telidon Behavioural Research*, *2*, Deptartment of Communications, May.

Whalen, T. and Mason, C. (1981) "The use of tree-structured index which contains three types of design defects" *Telidon Behavioural Research*, *2*, Deptartment of Communications, May.

Wilkinson, W. (1980) "Viewdata: The Prestel System" in *Videotext: the coming revolution in home/office information retrieval*, edited by Sigel, E., pp 57-86. Harmony Books, New York.

Williams, G. (1983) "The Lisa computer system" *Byte*, *8* (2), February.

Witten, I.H. (1982) "An interactive computer terminal interface which predicts user entries" Research report No. 82/84/3, Department of Computer Science, University of Calgary.

Witten, I.H., Cleary, J.G., and Darragh, J.J. (1983) "The reactive keyboard: A new techonlogy for text entry" *Proc Canadian Information Processing Conference*, Ottawa, Ontario, May.

Witten, I.H., Greenberg, S., and Cleary, J. (1983) "Personalizable directories: a case study in automatic user modelling" *Proc Graphics Interface 83*, Edmonton, Alberta, May.

Witten, I.H., Cleary, J., and Greenberg, S. (in press) "On frequency-based menu-splitting algorithms" *CIPS Conference*, Calgary, May.

Zipf, G.K. (1949) "Human behaviour and the principle of least effort" Ontario.

All quotes introducing chapters and sections that are not mentioned in the above list are from *The Oxford Dictionary of Quotations, 3rd Edition*, The Oxford University Press (1979).

Appendix 1. Comparison Of Menu Displays For Ordered Lists

Menu schemes provide a simple way for naive and casual users to interact with a system, using only numeric keypads (eg Cuff, 1980; Martin, 1973). Videotext and other textual databases systems (such as Zog; Robertson, *et al*, 1981) are normally built as hierarchical tree structures, where each menu item leads to subdivisions of "fields of knowledge" (Tompa, 1982). Several authors have undertaken investigations of different aspects of category menu representation, some of which were reported in section 2.5. Latremouille and Lee (1981) ran experiments testing the effect of adding descriptors to index items in Videotext, while Kiger's (1984) experiments studied the depth/breadth trade-off in a menu hierarchy. Engel *et al* (1983) proposed a novel method for redesigning category menus into a *map* which allows random access into a videotext database, another facet of the previously described *what, where and whence* system.

However, some information cannot easily be categorized on a semantic basis. We must consider the large, ordered, sequential lists present in the personalized telephone directory scheme. Although directories may sometimes be accessed semantically — through yellow pages — the personalization algorithm presupposes no such structure. In the absence of a natural *a priori* taxonomy, the only reasonable presentation scheme is to divide the name space into ranges using normal lexical ordering.

One study (Tombaugh and McEwen, 1982) compared taxonomic and alphabetic retrieval methods on Videotext. Their findings revealed little difference in mean search time and number of menu pages accessed between the two. However, a preference towards alphabetic displays was shown indirectly; users switched twice as often towards an alphabetic display when given a choice. But in general, little is known about the display of sequential ranges of menu items. Even the effect of menu length is unknown. Human factors experiments investigating the depth/breadth tradeoff in menu hierarchies have concluded that eight items is about the best length for taxonomic menus (eg Kiger, 1984). If the user is searching for an optimal match between his mental construct and a menu item, this can be interpreted in terms of the well-known phenomenon of "chunking" in short-term memory (Miller, 1956); for the best match cannot be determined until all items have been examined. But this reasoning does not apply to menus whose items represent ordered ranges of a list.

Another concern — and the focus of this Appendix — is the manner in which the ranges are presented.

A.1 Alternative menu displays.

Dictionaries contain very long alphabetically-ordered lists. But people do not normally search these lists sequentially when searching for a word definition. First, the book is usually opened to the approximate position of the item in the text. Further search is then facilitated through reading the keyword at the top of the page, which describe the precise location of the current page in the list. The keyword on the left page supplies a lower range delimiter for that page, for it indicates the first, and alphabetically lowest, word in the list. Similarly, an upper range delimiter on the right page indicates the last alphabetically uppermost word in that page. Telephone books, on the other hand, supply the full range delimiter, as illustrated by "Restaurants — Roofing" appearing on a single page. Any of the above methods can be used in a computerized menu system to indicate what range of the list is covered by a menu item. The characteristics of displays using these delimiters are the subject of this study.

A *range* is defined as a segment of an ordered list bounded by a lower delimiter and an upper delimiter. When divided into ranges, numbered for menu selection, a list has the form

```
      . . .
n.    lower delimiter  —>  upper delimiter
n+1.  lower delimiter  —>  upper delimiter
n+2.  lower delimiter  —>  upper delimiter
      . . .
```

The ranges are disjoint, and ordered so that the upper delimiter of one precedes the lower delimiter of the next.

A user's selection process is based on his ability to identify the range indicated by a menu item. One method of menu presentation simply uses the above display (see also Table A-1a). However, it is evident that this contains at least two separate kinds of redundancy. In the first place, (almost) half the delimiters in the display are implicitly supplied by the neighboring ranges. In the second place, many of the individual delimiters could likely be truncated without sacrificing information relevant to the selection process.

(A-1a) Complete Delimiter			(A-1b) Root Delimiter		
1) Arbor	—>	Barney	1) A	—	Barn
2) Barrymore	—>	Dacker	2) Barr	—	Dac
3) Danby	—>	Estovitch	3) Dan	—	E
4) Farquar	—>	Kalmer	4) F	—	Kalmerr
5) Kalmerson	—>	Moreen	5) Kalmers	—	More
6) Moriarty	—>	Praleen	6) Mori	—	Pra
7) Proctor	—>	Sageen	7) Pro	—	Sage
8) Sagin	—>	Ulston	8) Sagi	—	Ul
9) Unger	—>	Zlotsky	9) Un	—	Z

Table A-1a & b: Full Range Delimiter

The first kind of redundancy follows from the fact that the upper delimiter of item *n* provides no information relevant to the selection process that is not contained in item *n*+1's lower delimiter. However, it may be that this redundant information helps the user to make his selection, certainly in the case where the name sought happens to be item *n*'s upper delimiter. One alternative for menu display is therefore to eliminate the upper-delimiter column entirely, except for the upper delimiter of the very last entry. This last entry is necessary for error recovery in the event that the desired selection is outside the current range. An alternative is to eliminate the lower-delimiter column instead, except for the first entry. This defines three types of menu displays:

- *full-range delimiter* — both upper and lower delimiters are shown for each menu entry (Table A-1)
- *lower-range delimiter* — only the lower delimiter is shown on a menu entry, the upper one being given implicitly by the next menu entry (Table A-2)
- *upper-range delimiter* — only the upper delimiter is shown on a menu entry, the lower one being implied by the previous menu entry (Table A-3).

The second kind of redundancy is that truncation is possible without losing information because only enough characters of each delimiter need be displayed to distinguish it from the preceding or following range. The truncation operation is not entirely trivial; indeed, dictionaries and telephone directories do not attempt it. When the two range-delimiters have a common prefix, that prefix plus one further character will suffice. However, complications arise when one delimiter is itself a prefix of the other. With the occasional addition of a string-terminating character (eg "."), these problems can be solved to give an unambiguous truncation operation. Using this, a delimiter can have one of two *truncation levels:*

(A-2a) Complete Delimiter	(A-2b) Root Delimiter
1) Arbor	1) A
2) Barrymore	2) Barr
3) Danby	3) Dan
4) Farquar	4) F
5) Kalmerson	5) Kalmers
6) Moriarty	6) Mori
7) Proctor	7) Pro
8) Sagin	8) Sagi
9) Unger	9) Un
—(Zlotsky)	—(Z)

Table A-2a & b: Lower Range Delimiter

(A-3a) Complete Delimiter	(A-3b) Root Delimiter
—(Arbor)	—(A)
1) Barney	1) Barn
2) Dacker	2) Dac
3) Estovitch	3) E
4) Kalmer	4) Kalmera
5) Moreen	5) More
6) Praleen	6) Pra
7) Sageen	7) Sage
8) Ulston	8) Ul
9) Zlotsky	9) Z

Table A-3a & b: Upper Range Delimiter

- ***complete delimiter*** — the complete word is used as a delimiter (Tables A-1a, A-2a, A-3a)
- ***root delimiter*** — the word is truncated as much as possible without sacrificing ordering information (Tables A-1b, A-2b, A-3b).

These two factors, taken together, give a total of six different display methods. The differences are certainly significant in terms of the total number of characters displayed. The most verbose display will show both delimiters in full, for each menu item; while the tersest will show one truncated delimiter instead. In an example menu with ordinary surnames, the number of characters displayed can differ by a factor of around four. It is by no means clear which method makes it easiest for the user to scan the menu quickly and accurately.

Indeed, the issue is further compounded when menu ***span*** is considered. The ***span*** is defined as the alphabetical segment displayed by the complete menu page. The root page usually covers a ***wide span***; such as the complete alphabet (Table A-4a). Progressive selections through the menu hierarchy leads to correspondingly ***narrow spans***, where only small alphabetic intervals are covered (Table A-4b). In wide spans, discrimination of ranges can usually be done by the first or second letter. Narrow spans have a high number of delimiters with common roots, implying that a large number of letters must be used to allow discrimination (Table A-4). The greater complexity of narrow span menus implies that user performance may deteriorate markedly upon progression through deeper layers of the tree hierarchy.

Pilot experiments are designed and run to attempt both to quantify any differences between the six displays in terms of scanning speed and error rate, and to test qualitative differences of user preference. Computer experience of the user and menu spans are also studied. The tested null hypothesis is

(A-4a) Wide Span			(A-4b) Narrow Span		
1) Arbor	—>	Dacker	1) Mayland	—	Mazury
2) Danby	—>	Kalmer	2) McAdam	—	McAlden
3) Kalmerson	—>	Praleen	3) McAlpin	—	McArter
4) Proctor	—>	Ulston	4) McBean	—	McBeath
5) Unger	—>	Zlotsky	5) McBeave	—	McCabe

Table A-4a & b: Wide and Narrow menu spans

- Six menu display systems based on combinations of truncation and delimiter methods do not differ significantly from each other in terms of user scanning speed and error rate.
- Menu span has no significant effect on user scanning speed and error rate.
- User experience has no significant effect on user scanning speed and error rate.

A.2 Method.

Subjects. The subjects are forty-eight paid volunteers (university students). Half of them are solicited from senior computer science courses, and are labeled as "computer experts". The remainder are obtained from a junior computer science course intended for students majoring in other areas, and are labeled "computer novices".

Subject use. The experiment is a 3-level (range delimiter) by 2-level (truncation) by 2-level (computer experience) by 2-level (menu span) mixed factorial design (Figure A-1). Each subject is assigned to both levels of truncation and width, using counterbalanced ordering to overcome transfer effects. Half of the subjects are computer experts and half are novices, giving a total of 8 subjects per cell.

Apparatus. A Corvus Concept microcomputer connected to a VAX-11/780 is used to display all material for the experiment on a high resolution bit-mapped screen. Keystrokes are timed locally on the Corvus so that precise measurements can be made (to within 50 msec). Instructions to subjects are given first verbally and then on-line.

Design. Subjects are randomly assigned to one of the three *range delimiter* groups. Each subject of each group is exposed to two sets of menu displays employing different truncation methods. Each truncation level alternates wide and narrow span menus.

Procedure. Each set presents instructions on how to use that particular menu, followed by a practice session of twenty-five trials and a test session of eighty trials. Each trial comprises

		Truncated		Not Truncated	
		Narrow	Wide	Narrow	Wide
Full	Novice	S1-8	S1-8	S1-8	S1-8
	Expert	S9-16	S9-16	S9-16	S9-16
Upper	Novice	S17-24	S17-24	S17-24	S17-24
	Expert	S25-32	S25-32	S25-32	S25-32
Lower	Novice	S33-40	S33-40	S33-40	S33-40
	Expert	S40-48	S40-48	S40-48	S40-48

Figure A-1: Mixed Factor Anova Design

three parts.

- A name is presented to the user, who reads it and presses <return>. Timing begins.
- A menu display appears below the name and is scanned by the user to identify the range which contains the name.
- The user types the number of the appropriate menu item followed by <return>. Timing ends.

Although the menus are originally generated randomly, they are the same across all subjects.

Measures. The dependent measures are scanning time and error rate. The independent measures are the truncation and range-delimiter methods, menu span and experience of subject. Scanning time is the time in seconds between the appearance of the menu and the user's selection. Any selection which did not contain the indicated name is considered to be an error. Both scanning time and error rate are collected as averages after all the trials. In addition, for qualitative comparison with the quantitative results, a forced-choice decision about preferred truncation style is solicited.

Motivation. It is desirable to keep the subject's motivation level constant to avoid uncontrolled variation within a subject. A cash prize is therefore awarded to the subject with the best performance, where performance is a combination of error rate and scanning time.

A.3 Results.

Scanning speed and error rate were analyzed independently through use of the analysis of variance statistical package (P4V) supplied by BMDP statistical software (Dixon *et al*, 1981). All statistically significant results are significant at or beyond the 0.05 level. A qualitative questionnaire on truncation revealed no overall subject preference for either method.

A.3.1 Scanning Speed

A 3×2×2×2 analysis of variance (Kirk, 1968) on scanning speed revealed a significant main effect for range delimiters, experience and menu span, and an interaction between truncation and menu span (Table A-5).

Post-hoc comparisons among cells of the interaction matrix, using tests of simple main effects (Kirk, 1968), showed significant differences between menu span at both levels of truncation, whereas truncation was not significant at either level of menu span. Figure A-2 illustrates the magnitude of the menu span/truncation interaction. The nature of this interaction — the small effect of truncation contrasted to the large effect of span — allows us, for all practical purposes, to ignore it.

Contrast comparisons (Kirk, 1968) between the different levels of range delimiters showed significant differences between full and lower delimiters, full and upper delimiters, but none between lower and upper. These results are illustrated in Table A-6, which supplies the differences among cell means of the range delimiter levels.

Individual cell means of the significant main effects are supplied in Table A-7. Of note here is that all significant results on main effects have a difference between levels of at least one second.

A.3.2 Error Rate

A 3×2×2×2 analysis of variance (Kirk, 1968) on error rate revealed a significant main effect for range delimiters, experience and menu span, and interactions between range with experience, and range with span (Table A-8).

Post-hoc comparisons among cells of the range by experience interaction matrix , using tests of simple main effects (Kirk, 1968), showed significant differences for experience only at the lower range delimiters. Figure A-3 graphically illustrates the experience/range delimiter interaction, in which novices have a much greater error rate than experts at the lower range delimiter menu only.

Source	Degrees Freedom	Sums of Squares	Mean Squares	F-Ratio
Between				
Range Delimiter (R)	2, 42	65.1550	32.5775	3.23*
Experience (E)	1, 42	55.6066	55.6066	5.52*
R X E	2, 42	20.9023	10.4512	1.04
Subj. w.groups		423.1403	10.0748	
Within				
Truncation (T)	1, 42	0.3679	0.3679	0.44
T X R	2, 42	0.1022	0.0511	0.06
T X E	1, 42	1.3788	1.3788	1.66
T X R X E	2, 42	0.0505	0.0252	0.03
T X subj. w.groups		34.9044	0.8311	
Menu Span (S)	1, 42	79.3989	79.3989	216.04**
S X R	2, 42	2.1857	1.0928	2.97
S X E	1, 42	0.3651	0.3651	0.99
S X R X E	2, 42	0.7858	0.3929	1.07
S X subj. w.groups		15.4360	0.3675	
T X S	1, 42	1.0518	1.0518	14.78**
T X S X R	2, 42	0.1367	0.0683	0.96
T X S X E	1, 42	0.0188	0.0188	0.27
T X S X R X E	2, 42	0.1161	0.0580	0.82
T X S X subj. w.groups		423.1403	10.0748	

* $p < .05$
**$p < .01$

Table A-5: Anova Summary Table — Scanning Speed

Similarly, post-hoc comparisons among cells of the range by span interaction matrix , using tests of simple main effects (Kirk, 1968), showed span significant at all levels of range delimiters, but range delimiters significant only at the narrow span level. Figure A-4 graphically illustrates the range delimiter/span interaction, in which error rate is sensitive to delimiter display type at narrow span menus only, with lower range delimiters having the greatest error rate.

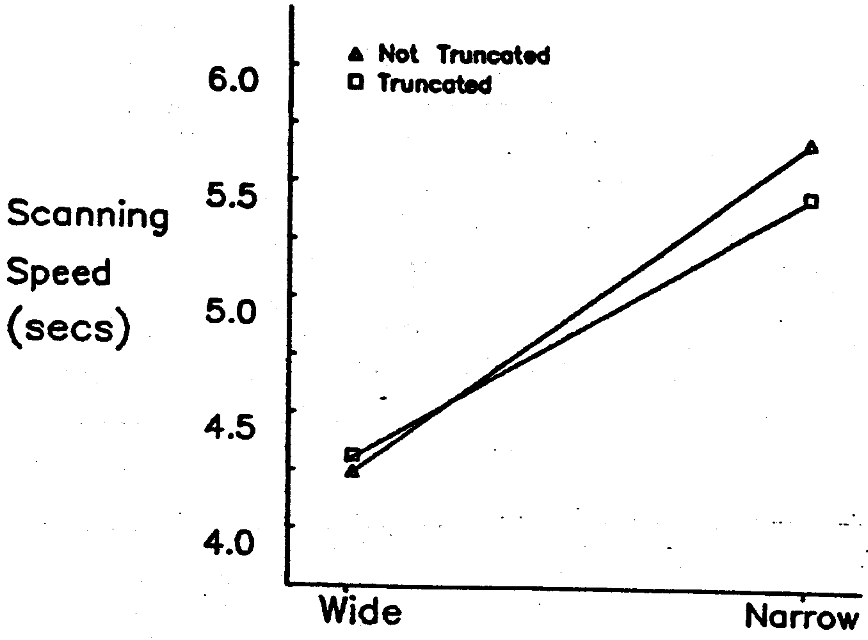

Figure A-2: Truncation/Span Interaction — Scanning Speed vs. Span

(means)	Full	Lower	Upper
Full = 5.72	—	1.15*	1.31*
Lower = 4.57		—	0.16
Upper = 4.41			—

* p < .05

Table A-6: Difference of range delimiter means (seconds)

A.4 Discussion

This study examines six menu displays — created by a combination of truncation and delimiter techniques — at different levels of user experience and menu span. The null hypothesis presented earlier states that display type, user experience and menu span have no effect on any of the measured variables. The test for the null hypothesis for truncation — that truncation has no effect on scanning speed or error rate — provides no evidence to support its

Factor	Level	Mean (Secs)	Error (%)
Range Delimiter	Full	5.72	8.125
	Upper	4.41	6.055
	Lower	4.57	10.660
Experience	Novice	5.44	9.922
	Expert	4.36	6.641
Span	Wide	4.25	4.219
	Narrow	5.54	12.340
Overall		4.9	8.281

Table A-7: Cell means of significant main effects and their levels

rejection. However, all other main effects and some two-way interactions support rejection of the null hypotheses. The results suggest the following points:

1. Full range delimiters are slower in subject scanning speed than either lower or upper delimiters (Table A-7).
2. Lower range delimiters have more errors per subject than either full or upper delimiters in a narrow span menu (Figure A-4).
3. Novices have more errors at the lower range delimiter than at full or upper delimiters (Figure A-3).
4. Narrow span menus are slower for subjects to scan than wide span menus (Table A-7).
5. Novices have slower scanning speeds than experts (Table A-7).

A.5 Conclusions

The results of this experiment supply clear guidelines. The upper range delimiter is preferable to any other kind, for it is faster than the full delimiter and less error prone than the lower delimiter. This is not surprising. Full range delimiters suffer from an over-redundancy of information which inhibits search efficiency. Lower range delimiters promote a bad search strategy, because an extra backup step exists. The correct choice precedes the menu entry which is alphabetically greater than the target. Neither the backup step nor the redundancy is present in an upper range delimiter display.

Source	Degrees Freedom	Sums of Squares	Mean Squares	F-Ratio
Between				
Range Delimiter (R)	2, 42	682.2270	341.1130	3.68*
Experience (E)	1, 42	516.7970	516.7970	5.57*
R × E	2, 42	869.7270	434.8630	4.69*
Subj. w.groups		3895.3125	92.7456	
Within				
Truncation (T)	1, 42	33.3333	33.3333	2.70
T × R	2, 42	27.4089	13.7044	1.11
T × E	1, 42	15.7552	15.7522	1.28
T × R × E	2, 42	11.7839	5.8919	0.48
T × subj. w.groups		517.9688	12.3326	
Menu Span (S)	1, 42	3168.7500	3168.7500	77.86**
S × R	2, 42	450.5860	225.2930	5.54**
S × E	1, 42	109.5050	109.5050	2.69
S × R × E	2, 42	118.0340	59.0169	1.45
S × subj. w.groups		1709.3750	40.6994	
T × S	1, 42	33.3333	33.3333	1.88
T × S × R	2, 42	56.7057	28.3529	1.60
T × S × E	1, 42	3.2552	3.2552	0.18
T × S × R × E	2, 42	4.3620	2.1800	0.12
T × S × subj. w.groups		746.0938	17.7641	

* $p < .05$
**$p < .01$

Table A-8: Anova Summary Table — Error Rate

Truncation techniques do not affect speed or errors. This mildly surprising fact is probably due to the user scanning only the first few letters of each word, rather than the complete chunk. In addition, the equal split between subjects over truncation preference promote the conclusion of truncation being of little relevance. Thus the selection of truncation method is at the discretion of the designer.

As expected, experienced users are more efficient in almost all measures than novices. But this experiment used general computer experience to differentiate novices and experts. No user had prior experience with any of the menu displays. Perhaps expert computer users are more confident with computer interactions and more adaptable to different systems. Or perhaps they are merely better typists. Whatever the cause, it would obviously be a mistake to disregard computer experience in choice of menu display.

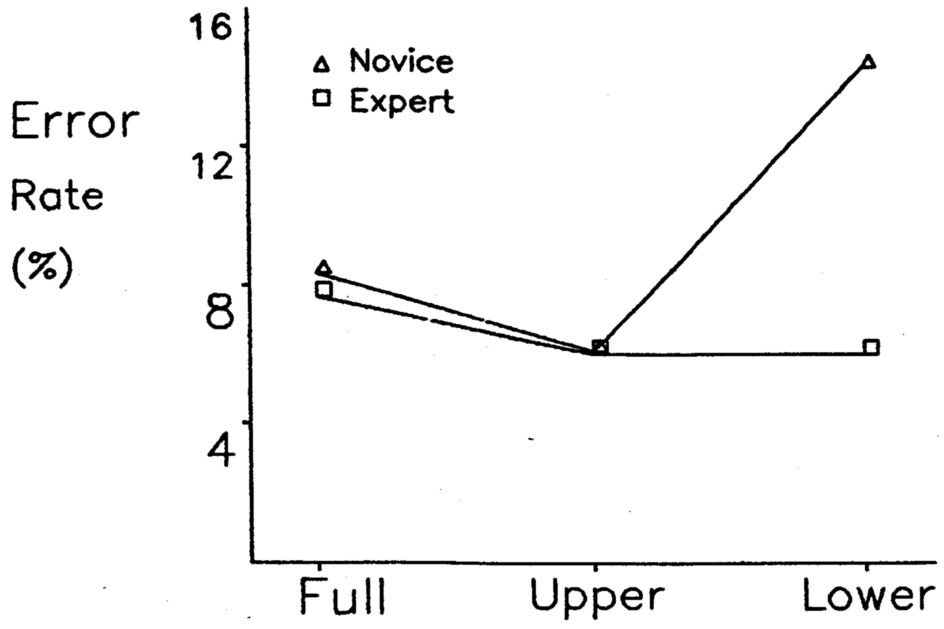

Figure A-3: Range/Experience Interaction —Error Rate vs. Range Delimiters

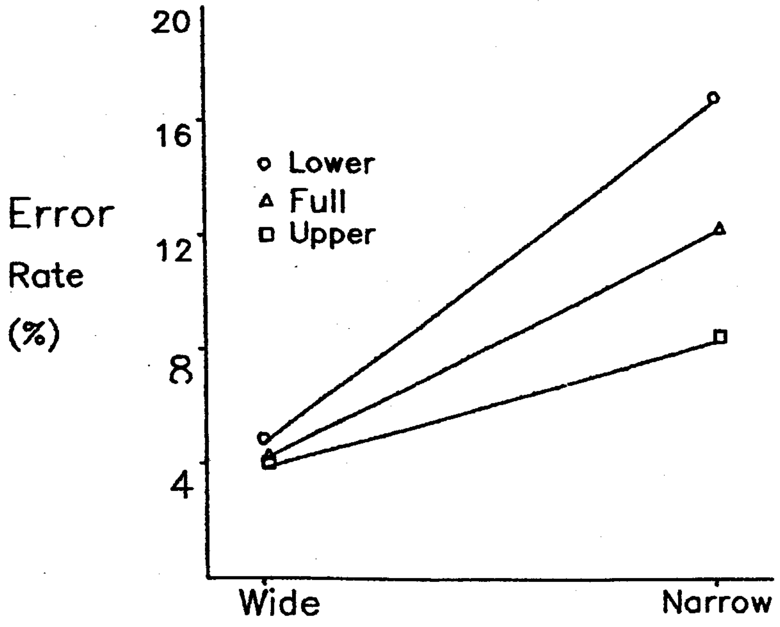

Figure A-4: Range/Span Interaction —Error Rate vs. Span

Narrow span menus are poorer in all measures than wide span ones. Menu span simulates the difference between root pages (wide span) and pages deep down the menu hierarchy (narrow span). Therefore we conclude that human performance degrades with tree depth in an alphabetic menu hierarchy.

Significant difference between means, whether it be error rate or scanning speed, is of no consequence if the magnitude of difference is small in relation to the task at hand. For example, scanning speed differences in menu types only slightly exceeds one second, a seemingly small amount. However, this represents a difference of approximately 25% in the overall selection time! The findings also indicate that proper selection of menu styles and strategies can reduce the incidence of errors by half. In real applications, an error in menu choice is associated with a high time and frustration penalty, for the user must backtrack to the previous menu. Clearly, the magnitude of difference in both scanning speed and error rate is important.

The final conclusion is simple. Upper delimiter menus are recommended with truncation at the discretion of the designer. Designers should expect novices to have slower scanning speeds and greater sensitivity to menu displays than experts. And finally, traversal of menu hierarchies should avoid, as much as possible, descending deep into the tree; otherwise user efficiency deteriorates.

Appendix B. Instructions for the Experiments

Instructions given to subjects in the two experiments reported in this thesis are described in this Appendix. The instructions consist of two parts: first a verbal summary given by the experimentor, followed by on-line instructions which subjects could peruse at their leisure. Most instructions are displayed in strict sequential order, but selected portions could be recalled at any time when the subject pressed a *help* key.

B.1 Instructions for Comparison of Six Menu Displays

Three screens of text instruct the user on the mechanics of the experiment. The fourth screen describes how one of the six menu display ranges operate, and can be redisplayed during the practice session by pressing the *help* button. It appears automatically at the beginning of the first and second session. Only one menu display instruction screen is included here — the others are quite similar. The fifth screen appears between the practice and real trials, while the sixth is displayed between the sessions.

Screen 1:

You will be given two different types of menus.
Each menu illustrates a different way of alphabetically dividing
people's names into ranges.

You will be given a name.
Your task is to select as quickly as possible the menu selection
which contains that name in its range.

Screen 2:

Outline of the experiment

Menu system 1
- -instructions
- -25 practice trials
- -80 real trials

Menu system 2
- -instructions
- -25 practice trials
- -80 real trials

For each menu type you will be given:
1) Instructions on how the menu works
2) A practice session in which any errors are pointed out to you
3) A real session which does not indicate if errors were made

For each trial:
A) You will be given a name.
B) WHEN YOU HAVE READ THE NAME, HIT <return> AND THE MENU WILL APPEAR.
C) FIND the name as quickly as possible without errors.

If you do make a mistake, just keep going.

Screen 3:

Finally, the HELP BUTTON will give you a description of how the current menu works.
However, this only works during the PRACTICE sessions.
Press it instead of a menu number.

If you have any questions, ask them NOW or during the practice sessions.............

Screen 4:

<—————— INSTRUCTIONS: Session #1 ——————>

This list is ordered in the following fashion:

1) Arbor	(From Arbor up to and not including Barrymore)
2) Barrymore	
3) Danby	
4) Farquar	
5) Kalmerson	
6) Moriarty	(From Moriarty up to and not including Proctor)
7) Proctor	
8) Sagin	
9) Unger	(From Unger up to and including Zlotsky)
—(Zlotsky)	

Example:	Jotterand	would be in selection 4.
	Liblong	would be in selection 5.
	Proctor	would be in selection 7.

Screen 5:

End of practice session.
Remember, this is it! Go quickly but try to avoid errors.
The $30. prize will be won by the person who has the fastest times and the least errors.

Screen 6:

Session #1 is over.
Session #2 will now begin, starting with a description followed by the practice session.

B.2 Instructions for Comparison of Directory Systems

Five screens of text instruct the user on the mechanics of the experiment. The sixth and seventh appear at the start of the dynamic and static sessions respectively. The eighth screen appears in between the practice and real trials, while the ninth is displayed between the sessions.

Screen 1:

OVERVIEW

This experiment will test you on two different techniques for going through menu hierarchies.

An example of a menu hierarchy representing a telephone directory is shown below. The user is searching for the name KING.

```
           — (Abbot)
Top        1) George        -Selection 1 contains all names
Level      2) Russel         BETWEEN Abbot and George.
Menu       3) Taber
           4) Zelnic        -Selection 2 contains all names
                             AFTER George up to Russel.
           Select: <2>


                            -Successive menus
           — (Gerry)         narrow the range down as much as
           1) King           possible
2nd        2) Niven
Level      3) Opel          -Note: Although King appears on this
Menu       4) Russel         menu, it is not a unique entry.
                             (see next menu)
           Select: <1>


                            -Finally, unique names start to appear.
           — (Gerry)         When the correct one is selected, you
           1) Gunther        have succeeded.
3rd        2) Hart, Martin
Level      3) Kant          -A unique name is indicated by the extra
Menu       4) King, Ralph    information supplied. Only unique
                             names have the first name and/or addresses
           Select: <4>       next to it.
```

Screen 2:

You will be tested on two different techniques for going through menu hierarchies, called STATIC and DYNAMIC.

1) A STATIC hierarchy remains the same all the time.
 -For example, if I look for the name King, I may go through a chain of menu selections such as: 2,1,3.
 -Successive accesses to King will have exactly the same chain of menu selections i.e. 2,1,3.

2) A DYNAMIC hierarchy changes from trial to trial.
 -Successive accesses to King will have different chains of menu selections:
 i.e.

-The next one may be	3,1,2
The third -	4,3
The fourth -	2

 -In addition, UNIQUE entries may appear at any level of the menu hierarchy.

Screen 3:

Outline Of The Experiment

SESSION 1

8 Practice trials
30 Real trials

SESSION 2

8 Practice trials
30 Real trials

One of the sessions uses STATIC menus, whereas the other uses DYNAMIC ones.

-During a single trial, you will be given a name.
Your task is to go down the menu hierarchy until the UNIQUE name is selected, going as fast as you can and making as few mistakes as possible.

-If you make a mistake, the system will always tell you.
You can then try again.

Screen 4:

A Single Menu Display Works In The Following Fashion:

—(Arbor)	
1) Barrymore	(From Arbor up to and including Barrymore)
2) Danby	
3) Farquar	
4) Kalmerson	(After Farquar up to and including Kalmerson)
5) Moriarty	
6) Proctor	
7) Sagin, George [Astronomy]	(Unique entry)
8) Unger	
9) Zlotsky	(After Unger up to and including Zlotsky)

Example:

Jotterand would be in selection 4.
Liblong would be in selection 5.
Proctor would be in selection 6.
Sagin is a unique entry.

Screen 5:

If you have any questions (which is probable at this point),
get the person running the experiment to answer them.
Some points to remember:

1) The prize will be won by the person with the best PERFORMANCE!!!
   If you make many mistakes, your performance is considered to be POOR.
   However, if you go very slowly, your performance is still POOR.
   You must go as fast as you can while making as few mistakes as possible.
2) You are only timed during the REAL sessions.
   Take your time during the practice session.
   If you have questions, ask them during the practice session
3) Your time starts when the first level menu in a trial appears.
   Your time ends when you have made a unique selection.
   (A success message is given to you).
   Thus you can take your time BETWEEN trials, but not within them!!!
   If a menu is in front of you, you're being timed!
4) ALL mistakes are counted.
   If you make a mistake, LOOK for the right entry. Don't guess.
5) A note will appear on the screen between the practice and real trials
   and between the main sessions.

Screen 6:

You will be given 8 practice trials which are not timed.
This is followed by 30 real trials which are timed.

The menu hierarchies in this session are DYNAMIC,
which means that there are no constant paths to an
entry.

Screen 7:

You will be given 8 practice trials which are not timed.
This is followed by 30 real trials which are timed.

The menu hierarchies in this session are STATIC,
which means that the path to a single entry is constant.
You may, if you wish, use your memory to remember
the path to an entry you have accessed before.

Screen 8:

End of practice session.
Remember, this is it! Go quickly but try to avoid errors.

The $30 prize will be won by the person who has the fastest times
and the least errors.

Reminder:
-If you see a menu in front of you, you are being timed!
-If there is no menu in front of you, you are not being timed.
This is the time to stretch out or ask questions.

GOOD LUCK!

Screen 9:

The first session is over.
The second session will now start.

Hit <carriage return> to start the next session: