HUMAN-COMPUTER INTERACTION, 1985, Volume 1, pp. 311-338

Direct Manipulation Interfaces

Edwin L. Hutchins, James D. Hollan, and
Donald A. Norman
University of California, San Diego

ABSTRACT

Direct manipulation has been lauded as a good form of interface design, and some interfaces that have this property have been well received by users. In this article we seek a cognitive account of both the advantages and disadvantages of direct manipulation interfaces. We identify two underlying phenomena that give rise to the feeling of directness. One deals with the information processing distance between the user's intentions and the facilities provided by the machine. Reduction of this distance makes the interface feel direct by reducing the effort required of the user to accomplish goals. The second phenomenon concerns the relation between the input and output vocabularies of the interface language. In particular, direct manipulation requires that the system provide representations of objects that behave as if they are the objects themselves. This provides the feeling of directness of manipulation.

A version of this paper also appears as a chapter in the book, *User Centered System Design: New Perspectives on Human-Computer Interaction* (Norman & Draper, 1986).

Authors' present address: Edwin L. Hutchins, James D. Hollan, and Donald A. Norman, Institute for Cognitive Science, University of California at San Diego, La Jolla, CA 92093.

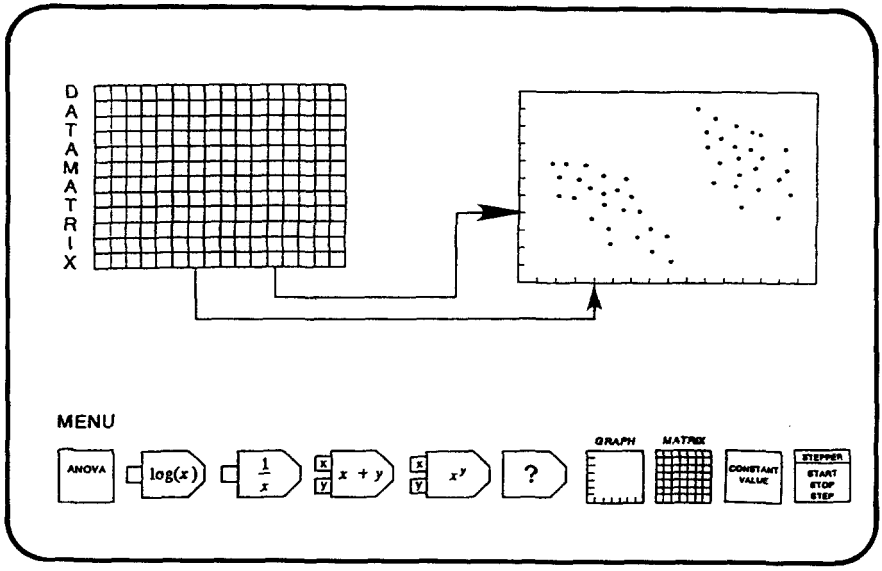
CONTENTS

1. DIRECT MANIPULATION

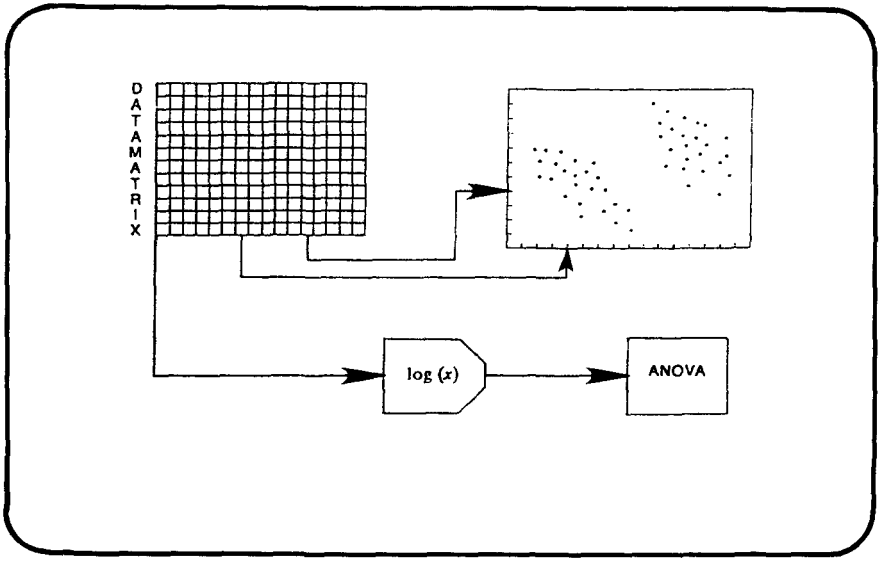
The best way to describe a direct manipulation interface is by example. Suppose we have a set of data to be analyzed with the numbers stored in matrix form. Their source and meaning are not important for this example. The numbers could be the output of a spreadsheet, a matrix of numerical values from the computations of a conventional programming language, or the results of an experiment. Our goal is to analyze the numbers, to see what relations exist among the rows and columns of the matrix. The matrix of numbers is represented on a computer display screen by an icon. To plot one column against another, simply get a copy of a graph icon, then draw a line from the output of one column to the x-axis input of the graph icon and another line from the output of the second column to the y-axis input (see Figure 1). Not what was wanted? Erase the lines and reconnect them. Want to see other graphs? Make more copies of the graph icons and connect them. Need a logarithmic transformation of one of the axes? Move up a function icon, type in the algebraic function that is desired ($y = \log x$, in this case) and connect it in the desired data stream. Want the analysis of variance of the logarithm of the data? Connect the matrix to the appropriate statistical icons. These examples are illustrated in Figure 1B.

Figure 1. An elementary example of doing simple statistical computations by direct manipulation. (A) The basic components: The data are contained in the matrix, represented by the icon in the upper left corner of the screen. At the bottom of the screen are basic icons that represent possible functions. To use one, a copy of the desired icon is moved to the screen and connected up, as is shown for the graph. (B) More complex interconnections, including the use of a logarithmic transformation of the data, a basic statistical package (for means and standard deviations), and an Analysis of Variance Package (ANOVA).

A



B



Now consider how we could partition the data. Suppose one result of our analysis was the scatter diagram shown in Figure 2. The straight line that has been fitted through the points is clearly inappropriate. The data fall into two quite different clusters and it would best to analyze each cluster separately. In the actual data matrix, the points that form the two clusters might be scattered randomly throughout the data set. The regularities are apparent only when we plot them. How do we pull out the clusters? Suppose we could simply circle the points of interest in the scatter plot and use each circled set as if it were a new matrix of values, each of which could be analyzed in standard ways, as shown in Figure 2B.

The examples of Figures 1 and 2 illustrate a powerful manipulation medium for computation. The promise of direct manipulation is that instead of an abstract computational medium, all the "programming" is done graphically, in a form that matches the way one thinks about the problem. The desired operations are performed simply by moving the appropriate icons onto the screen and connecting them together. Connecting the icons is the equivalent of writing a program or calling on a set of statistical subroutines, but with the advantage of being able to directly manipulate and interact with the data and the connections. There are no hidden operations, no syntax or command names to learn. What you see is what you get. Some classes of syntax errors are eliminated. For example, you can't point at a nonexistent object. The system requires expertise in the task domain, but only minimal knowledge of the computer or of computing.

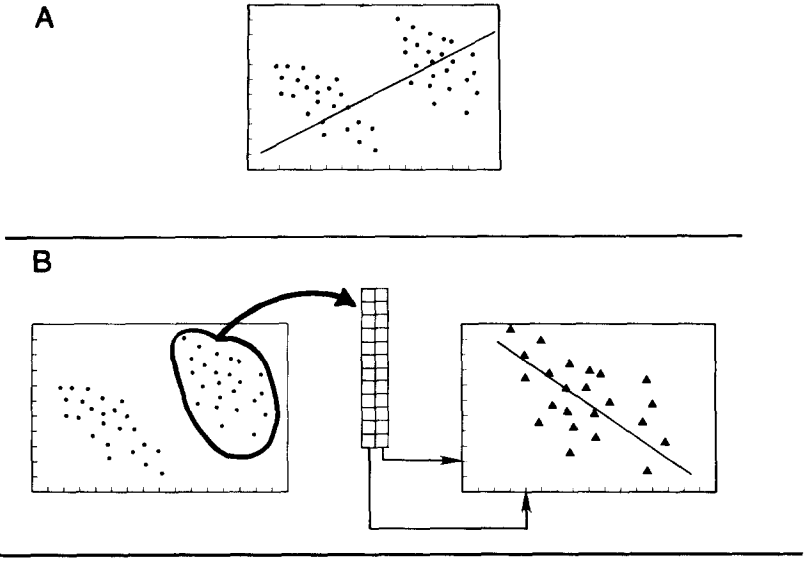
The term *direct manipulation* was coined by Shneiderman (1974, 1982, 1983) to refer to systems having the following properties:

1. Continuous representation of the object of interest.
2. Physical actions or labeled button presses instead of complex syntax.
3. Rapid incremental reversible operations whose impact on the object of interest is immediately visible. (Shneiderman, 1982, p. 251)

Direct manipulation interfaces seem remarkably powerful. Shneiderman (1982) has suggested that direct manipulation systems have the following virtues:

1. Novices can learn basic functionality quickly, usually through a demonstration by a more experienced user.
2. Experts can work extremely rapidly to carry out a wide range of tasks, even defining new functions and features.
3. Knowledgeable intermittent users can retain operational concepts.
4. Error messages are rarely needed.
5. Users can see immediately if their actions are furthering their goals, and if not, they can simply change the direction of their activity.

Figure 2. **(A) The scatter plot formed in Figure 1, along with the best fitting regression line to the data. It is clear that the data really fall into two quite distinct clusters and that it would be best to look at each independently. (B) The clusters are analyzed by circling the desired data, then treating the group of circled data as if they were a new matrix of values, which can be treated as a data source and analyzed in standard ways.**



6. Users have reduced anxiety because the system is comprehensible and because actions are so easily reversible. (Shneiderman, 1982, p. 251)

Can this really be true? Certainly there must be problems as well as benefits. It turns out that the concept of direct manipulation is complex. Moreover, although there are important benefits there are also costs. Like everything else, direct manipulation systems trade off one set of virtues and vices against another. It is important that we understand these trade-offs. A checklist of surface features is unlikely to capture the real sources of power in direct manipulation interfaces.

1.1. Early Examples of Direct Manipulation

Hints of direct manipulation programming environments have been around for quite some time. The first major landmark is Sutherland's *Sketchpad,* a graphical design program (Sutherland, 1963). Sutherland's goal was to devise a program that would make it possible for a person and a computer "to converse rapidly through the medium of line drawings." Sutherland's work is a land-

mark not only because of historical priority but because of the ideas that he helped develop: He was one of the first to discuss the power of graphical interfaces, the conception of a display as "sheets of paper," the use of pointing devices, the virtues of constraint representations, and the importance of depicting abstractions graphically.

Sutherland's ideas took 20 years to have widespread impact. The lag is perhaps due more to hardware limitations than anything else. Highly interactive, graphical programming requires the ready availability of considerable computational power, and it is only recently that machines capable of supporting this type of computational environment have become inexpensive enough to be generally available. Now we see these ideas in many of the computer-aided design and manufacturing systems, many of which can trace their heritage directly to Sutherland's work. Borning's *ThingLab* program (1979) explored a general programming environment, building upon many of Sutherland's ideas within the Smalltalk programming environment. More recently direct manipulation systems have been appearing with reasonable frequency. For example, *Bill Budge's Pinball Construction Set* (Budge, 1983) permits a user to construct an infinite variety of electronic pinball games by directly manipulating graphical objects that represent the components of the game surface. Other examples exist in the area of intelligent training systems (e.g., the Steamer system of Hollan, Hutchins, & Weitzman, 1984; Hollan, Stevens, & Williams, 1980). Steamer makes use of similar techniques and also provides tools for the construction of interactive graphical interfaces. Finally, spreadsheet programs incorporate many of the essential features of direct manipulation. In the lead article of *Scientific American*'s special issue on computer software, Kay (1984) claims that the development of dynamic spreadsheet systems gives strong hints that programming styles are in the offing that will make programming as it has been done for the past 40 years—that is, by composing text that represents instructions—obsolete.

1.2. The Goal: A Cognitive Account of Direct Manipulation

We see promise in the notion of direct manipulation, but as yet we see no explanation of it. There are systems with attractive features, and claims for the benefits of systems that give the user a certain sort of feeling, and even lists of properties that seem to be shared by systems that provide that feeling, but no account of how particular properties might produce the feeling of directness. The purpose of this article is to examine the underlying basis for direct manipulation systems. On the one hand, what is it that provides the feeling of "directness?" Why do direct manipulation systems feel so natural? What is so compelling about the notion? On the other hand, why can using such systems sometimes seem so tedious?

For us, the notion of "direct manipulation" is not a unitary concept, nor even something that can be quantified in itself. It is an orienting notion. "Directness" is an impression or a feeling about an interface. What we seek to do here is to characterize the space of interfaces and see where within that picture the range of phenomena that contribute to the feeling of directness might reside. The goal is to give cognitive accounts of these phenomena. At the root of our approach is the assumption that the feeling of directness results from the commitment of fewer cognitive resources. Or, put the other way around, the need to commit additional cognitive resources in the use of an interface leads to the feeling of indirectness. As we shall see, some of the production of the feeling of directness is due to adaptation by the user, so that the designer can neither completely control the process, nor take full credit for the feeling of directness that may be experienced by the user.

We will not attempt to set down hard and fast criteria under which an interface can be classified as direct or not direct. The sensation of directness is always relative; it is often due to the interaction of a number of factors. There are costs associated with every factor that increases the sensation of directness. At present we know of no way to measure the trade-off values, but we will attempt to provide a framework within which one can say what is being traded off against what.

2. TWO ASPECTS OF DIRECTNESS: DISTANCE AND ENGAGEMENT

There are two distinct aspects of the feeling of directness. One involves a notion of the distance between one's thoughts and the physical requirements of the system under use. A short distance means that the translation is simple and straightforward, that thoughts are readily translated into the physical actions required by the system and that the system output is in a form readily interpreted in terms of the goals of interest to the user. We will use the term *directness* to refer to the feeling that results from interaction with an interface. The term *distance* will be used to describe factors which underlie the generation of the feeling of directness.

The second aspect of directness concerns the qualitative feeling of engagement, the feeling that one is directly manipulating the objects of interest. There are two major metaphors for the nature of human-computer interaction, a conversation metaphor and a model-world metaphor. In a system built on the conversation metaphor, the interface is a language medium in which the user and system have a conversation about an assumed, but not explicitly represented world. In this case, the interface is an implied intermediary between the user and the world about which things are said. In a system built on the model-world metaphor, the interface is itself a world where the user can act,

and which changes state in response to user actions. The world of interest is explicitly represented and there is no intermediary between user and world. Appropriate use of the model-world metaphor can create the sensation in the user of acting upon the objects of the task domain themselves. We call this aspect of directness *direct engagement.*

2.1. Distance

We call one underlying aspect of directness *distance* to emphasize the fact that directness is never a property of the interface alone, but involves a relationship between the task the user has in mind and the way that task can be accomplished via the interface. Here the critical issues involve minimizing the effort required to bridge the gulf between the user's goals and the way they must be specified to the system.

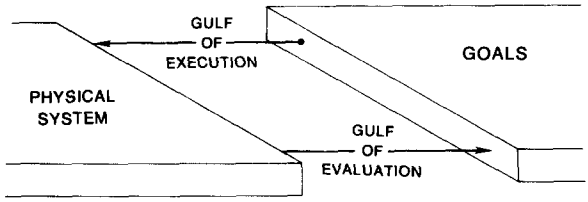
An interface introduces distance to the extent there are gulfs between a person's goals and knowledge and the level of description provided by the systems with which the person must deal. These are referred to as the *gulf of execution* and the *gulf of evaluation* (Figure 3). The gulf of execution is bridged by making the commands and mechanisms of the system match the thoughts and goals of the user. The gulf of evaluation is bridged by making the output displays present a good conceptual model of the system that is readily perceived, interpreted, and evaluated. The goal in both cases is to minimize cognitive effort.

We suggest that the feeling of directness is inversely proportional to the amount of cognitive effort it takes to manipulate and evaluate a system and, moreover, that cognitive effort is a direct result of the gulfs of execution and evaluation. The better the interface to a system helps bridge the gulfs, the less cognitive effort needed and the more direct the resulting feeling of interaction.

2.2. Direct Engagement

The description of the nature of interaction to this point begins to suggest how to make a system less difficult to use, but it misses an important point, a point that is the essence of direct manipulation. The analysis of the execution and evaluation process explains why there is difficulty in using a system, and it says something about what must be done to minimize the mental effort required to use a system. But there is more to it than that. The systems that best exemplify direct manipulation all give the *qualitative* feeling that one is directly engaged with control of the objects—not with the programs, not with the computer, but with the semantic objects of our goals and intentions. This is the feeling that Laurel (1986) discusses: a feeling of first-personness, of direct engagement with the objects that concern us. Are we analyzing data? Then we should be manipulating the data themselves; or if we are designing an analysis of data, we should be manipulating the analytic structures themselves. Are we

Figure 3. **The gulfs of execution and evaluation. Each gulf is unidirectional: The gulf of execution goes from goals to system state; the gulf of evaluation goes from system state to goals.**



playing a game? Then we should be manipulating directly the game world, touching and controlling the objects in that world, with the output of the system responding directly to our actions, and in a form compatible with them.

Historically, most interfaces have been built on the conversation metaphor. There is power in the abstractions that language provides (we discuss some of this later), but the implicit role of interface as an intermediary to a hidden world denies the user direct engagement with the objects of interest. Instead, the user is in direct contact with linguistic structures, structures that can be interpreted as referring to the objects of interest, but that are not those objects themselves. Making the central metaphor of the interface that of the model world supports the feeling of directness. Instead of describing the actions of interest, the user performs those actions. In a conventional interface, the system describes the results of the actions. In a model world the system directly presents the actions taken upon the objects. This change in central metaphor is made possible by relatively recent advances in technology. One of the exciting prospects for the study of direct manipulation is the exploration of the properties of systems that provide for direct engagement.

Building interfaces based on the model-world metaphor requires a special sort of relationship between the input interface language and the output interface language. In particular, the output language must represent its subject of discourse in a way that natural language does not normally do. The expressions of a direct manipulation output language must behave in such a way that the user can assume that they, in some sense, *are* the things they refer to. DiSessa (1985) calls this "naive realism." Furthermore, the nature of the relationship between input and output language must be such that an output expression can serve as a component of an input expression. Draper (1986) has coined the term *inter-referential* I/O to refer to relationships between input and output in which an expression in one can refer to an expression in the other. When these conditions are met, it is as if we are directly manipulating the things that the system represents.

Thus, consider a system in which a file is represented by an image on the screen and actions are done by pointing to and manipulating the image. In this

case, if we can specify a file by pointing at the screen representation, we have met the goal that an expression in the output language (in this case, an image) be allowed as a component of the input expression (in this case, by pointing at the screen representation). If we ask for a listing of files, we would want the result to be a representation that can, in turn, be used directly to specify the further operations to be done. Notice that this is not how a conversation works. In conversation, one may refer to what has been said previously, but one cannot operate upon what has been said. This requirement does not necessarily imply an interface of pictures, diagrams, or icons. It can be done with words and descriptions. The key properties are that the objects, whatever their form, have behaviors and can be referred to by other objects, and that referring to an object causes it to behave. In the file-listing example, we must be able to use the output expression that represents the file in question as a part of the input expression calling for whatever operation we desire upon that file, and the output expression that represents the file must change as a result of being referred to in this way. The goal is to permit the user to act as if the representation is the thing itself.

These conditions are met in many screen editors when the task is the arrangement of strings of characters. The characters appear as they are typed. They are then available for further operations. We treat them as though they are the things we are manipulating. These conditions are also met in the statistics example with which we opened this article (Figure 1), and in Steamer. The special conditions are not met in file-listing commands on most systems, the commands that allow one to display the names and attributes of file structure. The issue is that the outputs of these commands are simply "names" of the objects, and operating on the names does nothing to the objects to which the names refer. In a direct manipulation situation, we would feel that we had the files in front of us, that the program that "listed" the files actually placed the files before us. Any further operation on the files would take place upon the very objects delivered by the directory-listing command. This would provide the feeling of directly manipulating the objects that were returned.

The point is that when an interface presents a world of behaving objects rather than a language of description, manipulating a representation can have the same effects and the same feel as manipulating the thing being represented. The members of the audience of a well-staged play willfully suspend their beliefs that the players are actors and become directly engaged in the content of the drama. In a similar way, the user of a well-designed model-world interface can willfully suspend belief that the objects depicted are artifacts of some program and can thereby directly engage the world of the objects. This is the essence of the "first-personness" feeling of direct engagement. Let us now return to the issue of distance and explore the ways that an interface can be direct or indirect with respect to a particular task.

3. TWO FORMS OF DISTANCE: SEMANTIC AND ARTICULATORY

Whenever we interact with a device, we are using an interface language. That is, we must use a language to describe to the device the nature of the actions we wish to have performed. This is true regardless of whether we are dealing with an interface based on the conversation metaphor or on the model-world metaphor, although the properties of the language in the two cases are different. A description of desired actions is an expression in the interface language.

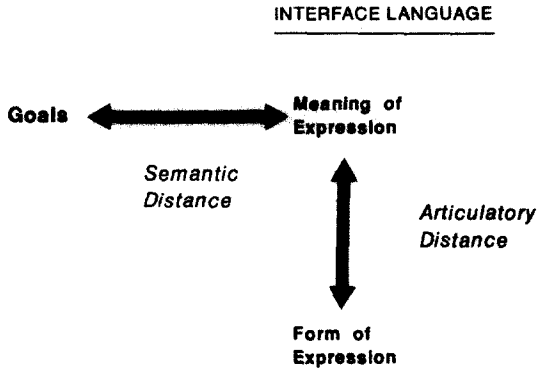
The notion of an interface language is not confined to the everyday meaning of language. Setting a switch or turning a steering wheel can be expressions in an interface language if switch setting or wheel turning are how one specifies the operations that are to be done. After an action has been performed, evaluation of the outcome requires that the device make available some indication of what has happened: that output is an expression in the output interface language. Output interface languages are often impoverished. Frequently the output interface language does not share vocabulary with the input interface language. Two forms of interface language—two dialects, if you will—must exist to span the gulfs between user and device: the input interface language and the output interface language.

Both the languages people speak and computer programming languages are almost entirely symbolic in the sense that there is an arbitrary relationship between the form of a vocabulary item and its meaning. The reference relationship is established by convention and must be learned. There is no way to infer meaning from form for most vocabulary items. Because of the relative independence of meaning and form we describe separately two properties of interface languages: semantic distance and articulatory distance. Figure 4 summarizes the relationship between semantic and articulatory distance. In the following sections we treat each of these distances separately and discuss them in relation to the gulfs of execution and evaluation.

3.1. Semantic Distance

Semantic distance concerns the relation of the meaning of an expression in the interface language to what the user wants to say. Two important questions about semantic distance are (1) *Is it possible to say what one wants to say in this language?* That is, does the language support the user's conception of the task domain? Does it encode the concepts and distinctions in the domain in the same way that the user thinks about them? (2) *Can the things of interest be said concisely?* Can the user say what is wanted in a straightforward fashion, or must the user

Figure 4. **Every expression in the interface language has a meaning and a form. Semantic distance reflects the relationship between the user intentions and the meaning of expressions in the interface languages both for input and output. Articulatory distance reflects the relationship between the physical form of an expression in the interaction language and its meaning, again, both for input and output. The easier it is to go from the form or appearance of the input or output to meaning, the smaller the articulatory distance.**



construct a complicated expression to do what appears in the user's thoughts as a conceptually simple piece of work?

Semantic distance is an issue with all languages. Natural languages generally evolve such that they have rich vocabularies for domains that are of importance to their speakers. When a person learns a new language—especially when the language is from a different culture—the new language may seem indirect, requiring complicated constructs to describe things the learner thinks should be easy to say. But the differences in apparent directness reflect differences in what things are thought important in the two cultures. Natural languages can and do change as the need arises. This occurs through the introduction of new vocabulary or by changing the meaning of existing terms. The result is to make the language semantically more direct with respect to the topic of interest.

3.2. Semantic Distance in the Gulfs of Execution and Evaluation

> Beware the Turing tar-pit in which everything is possible but nothing of interest is easy (Perlis, 1982, p. 10).

The Gulf of Execution

At the highest level of description, a task may be described by the user's intention: "compose this piece" or "format this paper." At the lowest level of description, the performance of the task consists of the shuffling of bits inside the machine. Between the interface and the low-level operations of the machine is

the system-provided task-support structure that implements the expressions in the interface language. The situation that Perlis (1982) called the "Turing tarpit" is one in which the interface language lies near or at the level of bit shuffling of a very simple abstract machine. In this case, the entire burden of spanning the gulf from user intention to bit manipulation is carried by the user. The relationship between the user's intention and the organization of the instructions given to the machine is distant, complicated, and hard to follow. Where the machine is of minimal complexity, as is the case with the Turing machine example, the wide gulf between user intention and machine instructions must be filled by the user's extensive planning and translation activities. These activities are difficult and rife with opportunities for error.

Semantic directness requires matching the level of description required by the interface language to the level at which the person thinks of the task. It is always the case that the user must generate some information-processing structure to span the gulf. Semantic distance in the gulf of execution reflects how much of the required structure is provided by the system and how much by the user. The more that the user must provide, the greater the distance to be bridged.

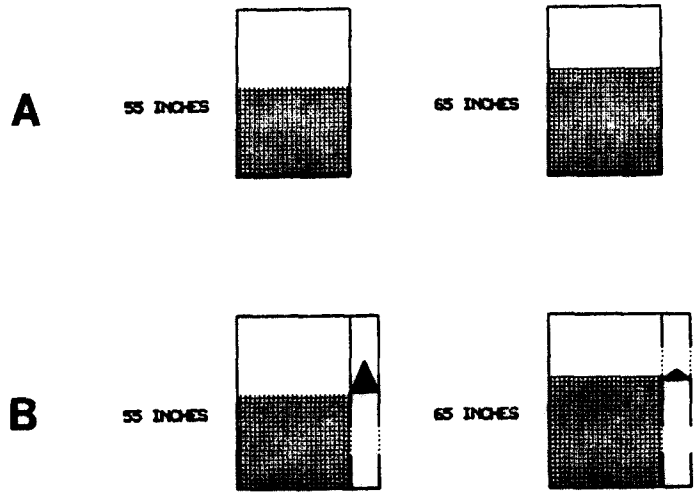
The Gulf of Evaluation

On the evaluation side, semantic distance refers to the amount of processing structure that is required for the user to determine whether the goal has been achieved. If the terms of the output are not those of the user's intention, the user will be required to translate the output into terms that are compatible with the intention in order to make the evaluation. For example, suppose a user's intent is to control how fast the water level in a tank rises. The user does some controlling action and observes the output. But if the output only shows the current value, the user has to observe the value over time and mentally compare the values at different times to see what the rate of change is (see Figure 5). The information needed for the evaluation is in the output, but it is not there in a form that directly fits the terms of the evaluation. The burden is on the user to perform the required transformations, and that requires effort. Suppose the rate of change were directly displayed, as in Figure 5B. This indication reduces the mental workload, making the semantic distance between intentions and output language much shorter.

3.3. Reducing the Semantic Distance That Must Be Spanned

Figure 5 provides one illustration of how semantic distance can be changed. In general, there are only two basic ways to reduce the distance, one from the system side (requiring effort on the part of the system designer), the other from the user side (requiring effort on the part of the user). Each direction of bridge building has several components. Here let us consider the following possibili-

Figure 5. **Matching user's intentions by appropriate output language. The user attempts to control the rate at which the water level in the tank is rising. In (A), the only indication is a meter that shows the current level. This requires the user to observe the meter over time and to do a mental computation on the observations. (B) shows a display that is more semantically direct: The rate of change is graphically indicated. (These illustrations are from the working Steamer system of Hollan, Hutchins, & Weitzman, 1984.)**



ties: (1) The designer can construct higher-level and specialized languages that move toward the user, making the semantics of the input and output languages match that of the user. (2) The user can develop competence by building new mental structures to bridge the gulfs. In particular, this requires the user to automate the response sequence and to learn to think in the same language as that required by the system.

Higher-Level Languages

One way to bridge the gulf between the intentions of the user and the specifications required by the computer is well known: Provide the user with a higher-level language, one that directly expresses frequently encountered structures of problem decomposition. Instead of requiring the complete decomposition of the task to low-level operations, let the task be described in the same language used within the task domain itself. Although the computer still requires low-level specification, the job of translating from the domain language to the programming language can be taken over by the machine itself.

This implies that designers of higher-level languages should consider how to develop interface languages for which it will be easy for the user to create the mediating structure between intentions and expressions in the language. One way to facilitate this process is to provide consistency across the interface sur-

face. That is, if the user builds a structure to make contact with some part of the interface surface, a savings in effort can be realized if it is possible to use all or part of that same structure to make contact with other areas.

The result of matching a language to the task domain brings both good news and bad news. The good news is that tasks are easier to specify. Even if considerable planning is still required to express a task in a high-level language, the amount of planning and translation that can be avoided by the user and passed off to the machine can be enormous. The bad news is that the language has lost generality. Tasks that do not easily decompose into the terms of the language may be difficult or impossible to represent. In the extreme case, what can be done is easy to do, but outside that specialized domain, nothing can be done.

The power of a specialized language system derives from carefully specified primitive operations, selected to match the predicted needs of the user, thus capturing frequently occurring structures of problem decomposition. The trouble is that there is a conflict between generality and matching to any specific problem domain. Some high-level languages and operating systems have attempted to close the gap between user intention and the interaction language while preserving freedom and ease of general expression by allowing for extensibility of the language or operating system. Such systems allow the users to move the interface closer to their conception of the task.

The Lisp language and the UNIX operating system serve as examples of this phenomenon. Lisp is a general-purpose language, but one that has extended itself to match a number of special high-level domains. As a result, Lisp can be thought of as having numerous levels on top of the underlying language kernel. There is a cost to this method. As more and more specialized domain levels get added, the language system gets larger and larger, becoming more clumsy to use, more expensive to support, and more difficult to learn. Just look at any of the manuals for the large Lisp systems (Interlisp, Zetalisp) to get a feel for the complexity involved. The same is true for the UNIX operating system, which started out with a number of low-level, general primitive operations. Users were allowed (and encouraged) to add their own, more specialized operations, or to package the primitives into higher-level operations. The results in all these cases are massive systems that are hard to learn and that require a large amount of support facilities. The documentation becomes huge, and not even system experts know all that is present. Moreover, the difficulty of maintaining such a large system increases the burden on everyone, and the possibility of having standard interfaces to each specialized function has long been given up.

The point is that as the interface approaches the user's intention end of the gulf, functions become more complicated and more specialized in purpose. Because of the incredible variety of human intentions, the lexicon of a language that aspires to both generality of coverage and domain-specific functions can grow very large. In any of the modern dialects of Lisp one sees a microcosm

of the argument about high-level languages in general. The fundamentals of the language are simple, but a great deal of effort is required to do anything useful at the low level of the language itself. Higher-level functions written in terms of lower-level ones make the system easier to use when the functions match intentions, but in doing so they may restrict possibilities, proliferate vocabulary, and require that a user know an increasing amount about the language of interaction rather than the domain of action.

Make the Output Show Semantic Concepts Directly

An example of reducing semantic distance on the output side is provided by the scenario of controlling the rate of filling a water tank, described in Figure 5. In that situation, the output display was modified to show rate of flow directly, something normally not displayed but instead left to the user to compute mentally.

In similar fashion, the change from line-oriented text editors to screen-oriented text editors, where the effects of editing commands can be seen instantly, is another example of matching the display to the user's semantics. In general, the development of WYSIWYG ("What You See Is What You Get") systems provides other examples. And finally, spreadsheet programs have been valuable, in part because their output format continually shows the state of the system as values are changed.

The attempt to develop good semantic matches with the system output confronts the same conflict between generality and power faced in the design of input languages. If the system is too specific and specialized, the output displays lack generality. If the system is too rich, the user has trouble learning and selecting among the possibilities. One solution for both the output and input problem is to abandon hope of maintaining general computing and output ability and to develop special-purpose systems for particular domains or tasks. In such a world, the location of the interface in semantic space is pushed closer to the domain language description. Here, things of interest are made simple because the lexicon of the interface language maps well into the lexicon of domain description. Considerable planning may still go on in the conception of the domain itself, but little or no planning or translation is required to get from the language of domain description to the language of the interface. The price paid for these advantages is a loss of generality: Many things are unnatural or even impossible.

Automated Behavior Does Not Reduce Semantic Distance

Cognitive effort is required to plan a sequence of actions to satisfy some intent. Generally, the more structure required of the user, the more effort use of the system will entail. However, this gap can be overcome if the users become familiar enough with the system. Structures that are used frequently need not

be rebuilt every time they are needed if they have been remembered. Thus, a user may remember how to do something rather than having to rederive how to do it. It is well known that when tasks are practiced sufficiently often, they become automated, requiring little or no conscious attention. As a result, over time the use of an interface to solve a particular set of problems will feel less difficult and more direct. Experienced users will sometimes argue that the interface they use directly satisfies their intentions, even when less skilled users complain of the complexity of the structures. To skilled users, the interface feels direct because the invocation of mediating structure has been automated. They have learned how to transform frequently arising intentions into action specifications. The result is a feeling of directness as compelling as that which results from semantic directness. As far as such users are concerned, the intention comes to mind and the action gets executed. There are no conscious intervening stages. (For example, a user of the vi text editor expressed this as follows: "I am an expert user of vi, and when I wish to delete a word, all I do is think 'delete that word,' my fingers automatically type 'dw,' and the word disappears from the screen. How could anything be more direct?")

The frequent use of even a poorly designed interface can sometimes result in a feeling of directness like that produced by a semantically direct interface. A user can compensate for the deficiencies of the interface through continual use and practice so that the ability to use it becomes automatic, requiring little conscious activity. While automatism is one factor which can contribute to a feeling of directness, it is essential for an interface designer to distinguish it from semantic distance. Automatization does not reduce the semantic distance that must be spanned; the gulfs between a user's intentions and the interface must still be bridged by the user. Although practice and the resulting expertise can make the crossing less difficult, it does not reduce the magnitude of the gulfs. Planning activity may be replaced by a single memory retrieval so that instead of figuring out what to do, the user remembers what to do. Automatization may feel like direct control, but it comes about for completely different reasons than semantic directness. Automatization is useful, for it improves the interaction of the user with the system, but the feeling of directness it produces depends only on how much practice a particular user has with the system and thus gives the system credit for the work the user has done. Although we need to remember that this happens, that users may adjust themselves to the interface and, with sufficient practice, may view it as directly supporting their intentions, we need to distinguish between the cases in which the feeling of directness originates from a close semantic coupling between intentions and the interface language and that which originates from practice. The resultant feeling of directness might be the same in the two cases, but there are crucial differences between how the feeling is acquired and what one needs to do as an interface designer to generate it.

The User Can Adapt to the System Representation

Another way to span the gulf is for the users to change their own conceptualization of the problem so that they come to think of it in the same terms as the system. In some sense, this means that the gulf is bridged by moving the user closer to the system. Because of their experience with the system, the users change both their understanding of the task and the language with which they think about issues. This is related to the notion of linguistic determinism. If it is true that the way we think about something is shaped by the vocabulary we have for talking about it, then it is important for the designer of a system to provide the user with a good representation of the task domain in question. The interface language should provide a powerful, productive way of thinking about the domain.

This form of the users adapting to the system representation takes place at a more fundamental level than the other ways of reducing semantic distance. While moving the interface closer to the users' intentions may make it difficult to realize some intentions, changing the users' conception of the domain may prevent some intentions from arising at all. So while a well-designed special-purpose language may give the users a powerful way of thinking about the domain, it may also restrict the users' flexibility to think about the domain in different ways.

The assumption that a user may change conceptual structure to match the interface language follows from the notion that every interface language implies a representation of the tasks it is applied to. The representation implied by an interface is not always a coherent one. Some interfaces provide a collection of partially overlapping views of a task domain. If a user is to move toward the model implied by the interface, and thus reduce the semantic distance, that model should be coherent and consistent over some conception of the domain. There is, of course, a trade-off here between the costs to the user of learning a new way to think about a domain and the potential added power of thinking about it in the new way.

Virtuosity and Semantic Distance

Sometimes users have a conception of a task and of a system that is broader and more powerful than that provided by an interface. The structures they build to make contact with the interface go beyond it. This is how we characterize virtuoso performances in which the user may "misuse" limited interface tools to satisfy intentions that even the system designer never anticipated. In such cases of virtuosity the notion of semantic distance becomes more complicated and we need to look very carefully at the task that is being accomplished. Semantic directness always involves the relationship between the task one wishes to accomplish and the ways the interface provides for accomplishing it.

If the task changes, then the semantic directness of the interface may also change.

Consider a musical example: Take the task of producing a middle-C note on two musical instruments, a piano and a violin. For this simple task, the piano provides the more direct interface because all one need do is find the key for middle-C and depress it, whereas on the violin, one must place the bow on the G string, place a choice of fingers in precisely the right location on that string, and draw the bow. A piano's keyboard is more semantically direct than the violin's strings and bow for the simple task of producing notes. The piano has a single well-defined vocabulary item for each of the notes within its range, while the violin has an infinity of vocabulary items, many of which do not produce proper notes at all. However, when the task is playing a musical piece well rather than simply producing notes, the directness of the interfaces can change. In this case, one might complain that a piano has a very indirect interface because it is a machine with which the performer "throws hammers at strings." The performer has no direct contact with the components that actually produce the sound, and so the production of desired nuances in sound is more difficult. Here, as musical virtuosity develops, the task that is to be accomplished also changes from just the production of notes to concern for how to control more subtle characteristics of the sounds like vibrato, the slight changes in pitch used to add expressiveness. For this task the violin provides a semantically more direct interface than the piano. Thus, as we have argued earlier, an analysis of the nature of the task being performed is essential in determining the semantic directness of an interface.

3.4. Articulatory Distance

In addition to its meaning, every vocabulary item in every language has a physical form and that form has an internal structure. Words in natural languages, for example, have phonetic structure when spoken and typographic structure when printed. Similarly, the vocabulary items that constitute an interface language have a physical structure. Where *semantic distance* has to do with the relationship between user's intentions and meanings of expressions, *articulatory distance* has to do with the relationship between the meanings of expressions and their physical form. On the input side, the form may be a sequence of character-selecting key presses for a command language interface, the movement of a mouse and the associated "mouse clicks" in a pointing device interface, or a phonetic string in a speech interface. On the output side, the form might be a string of characters, a change in an iconic representation, or variation in an auditory signal.

There are ways to design languages such that the relationships between the forms of the vocabulary items and their meanings are not arbitrary. One tech-

nique is to make the physical form of the vocabulary items structurally similar to their meanings. In spoken language this relationship is called onomatopoeia. Onomatopoetic words in spoken language refer to their meanings by imitating the sound they refer to. Thus we talk about the "boom" of explosions or the "cock-a-doodle-doo" of roosters. There is an economy here in that the user's knowledge of the structure of the surface acoustical form has a nonarbitrary relation to meaning. There is a directness of reference in this imitation; an intervening level of arbitrary symbolic relations is eliminated. Other uses of language exploit this effect partially. Thus, although the word "long" is arbitrarily associated with its meaning, sentences like "She stayed a looooooooooong time" exploit a structural similarity between the surface form of "long" (whether written or spoken) and the intended meaning. The same sorts of things can be done in the design of interface languages.

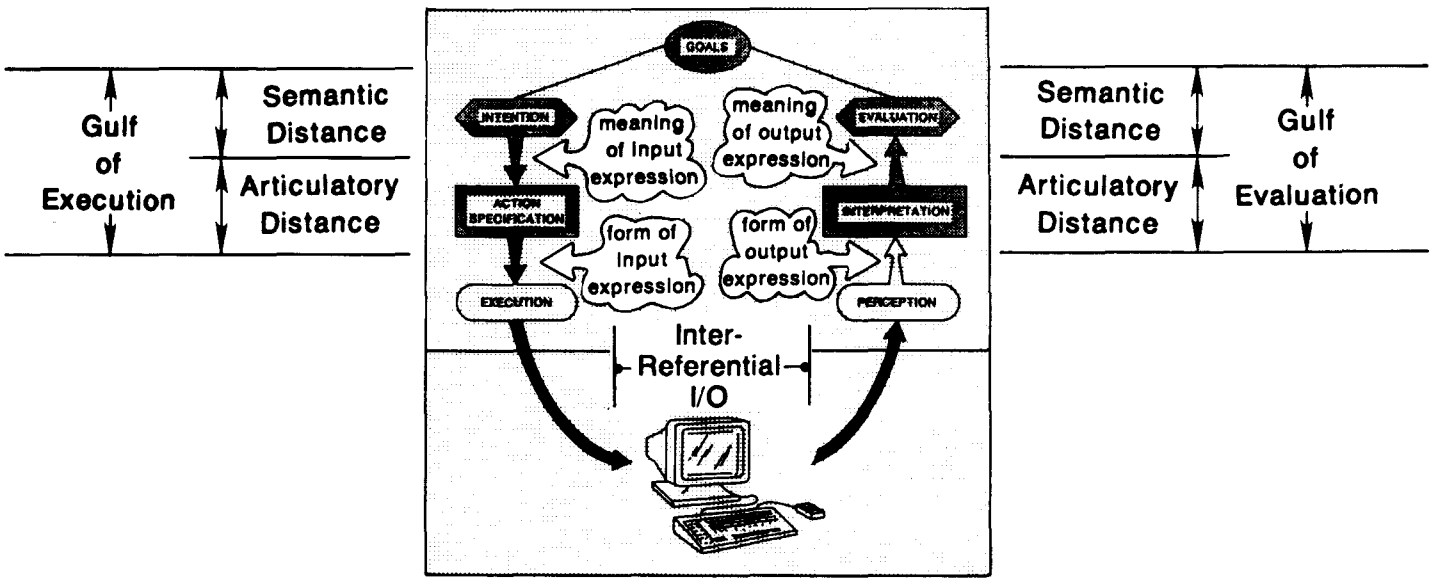
In many ways, the interface languages should have an easier time of exploiting articulatory similarity than do natural languages because of the rich technological base available to them. Thus, if the intent is to draw a diagram, the interface might accept as input drawing motions. In turn, it could present as output diagrams, graphs, and images. If one is talking about sound patterns in the input interface language, the output could be the sounds themselves. The computer has the potential to exploit articulatory similarities through technological innovation in the varieties of dimensions upon which it can operate. This potential has not been exploited, in part because of economic constraints. The restriction to simple keyboard input limits the form and structure of the input languages and the restriction to simple, alphanumeric terminals with small, low-resolution screens, limits the form and structure of the output languages.

3.5. Articulatory Distance in the Gulfs of Execution and Evaluation

The relationships among semantic distance, articulatory distance, and the gulfs of execution and evaluation are illustrated in Figure 6.

Take the simple, commonplace activity of moving a cursor on the screen. If we do this by moving a mouse, pointing with a finger or a light pen at the screen, or otherwise mimicking the desired motion, then at the level of action execution, these interactions all exhibit articulatory directness. The meaning of the intention is cursor movement and the action is specified by means of a similar movement. One way to achieve articulatory directness at the input side is to provide an interface that permits specification of an action by mimicking it, thus supporting an articulatory similarity between the vocabulary item and its meaning. Any nonarbitrary relationship between the form of an item and its meaning can be a basis for articulatory directness. While structural relationships of form to meaning may be desirable, it is sometimes necessary to re-

Figure 6. Forming an intention is the activity that spans semantic distance in the gulf of execution. The intention specifies the meaning of the input expression that is to satisfy the user's goal. Forming an action specification is the activity that spans articulatory distance in the gulf of execution. The action specification prescribes the form of an input expression having the desired meaning. The form of the input expression is executed by the user on the machine interface and the form of the output expression appears on the machine interface, to be perceived by the user. When some part of the form of a previous output expression is incorporated in the form of a new input expression, the input and output are said to be inter-referential. Interpretation is the activity that spans articulatory distance in the gulf of evaluation. Interpretation determines the meaning of the output expression from the form of the output expression. Evaluation is the activity that spans semantic distance in the gulf of evaluation. Evaluation assesses the relationship between the meaning of the output expression and the user's goal.



sort to an arbitrary relationship of form to meaning. Still, some arbitrary relationships are easier to learn than others. It may be possible to exploit previous user knowledge in creating this relationship. Much of the work on command names in command language interfaces is an instance of trying to develop memorable and discriminable relationships between the forms and the meanings of command names (Black & Moran, 1982; Black & Sebrechts, 1981; Carrol, 1985).

Articulatory directness on the output side is similar. If the user is following the changes in some variable, a moving graphical display can provide articulatory directness. A table of numbers, although containing the same semantic information, does not provide articulatory directness. Thus, the graphical display and the table of numbers might be equal in semantic directness, but unequal in articulatory directness. The goal of designing for articulatory directness is to couple the perceived form of action and meaning so naturally that the relationships between intentions and actions and between actions and output seem straightforward and obvious.

In general, articulatory directness is highly dependent upon I/O technology. Increasing the articulatory directness of actions and displays requires a much richer set of input/output devices than most systems currently have. In addition to keyboards and bit-mapped screens, we see the need for various forms of pointing devices. Such pointing devices have important *spatio-mimetic* properties and thus support the articulatory directness of input for tasks that can be represented spatially. The mouse is useful for a wide variety of tasks not because of any properties inherent in itself, but because we map so many kinds of relationships (even ones that are not intrinsically spatial) on to spatial metaphors. In addition, there are often needs for sound and speech, certainly as outputs, and possibly as inputs. Precise control of timing will be necessary for those applications where the domain of interest is time sensitive. Perhaps it is stretching the imagination beyond its willing limits, but Galton (1894) suggested and carried out a set of experiments on doing arithmetic by sense of smell. Less fancifully conceived, input might be sensitive not only to touch, place, and timing, but also to pressure or to torque (see Buxton, 1986; Minsky, 1984).

4. DIRECT ENGAGEMENT

Direct engagement occurs when a user experiences direct interaction with the objects in a domain. Here there is a feeling of involvement directly with a world of objects rather than of communication with an intermediary. The interactions are much like interacting with objects in the physical world. Actions apply to the objects, observations are made directly upon those objects, and the interface and the computer become invisible. Although we believe this feeling of direct engagement to be of critical importance, in fact, we know little about

the actual requirements for producing it. Laurel (1986) discusses some of the requirements. At a minimum, to allow a feeling of direct engagement the system requires the following:

> Execution and evaluation should exhibit both semantic and articulatory directness.
>
> Input and output languages of the interface should be inter-referential, allowing an input expression to incorporate or make use of a previous output expression. This is crucial for creating the illusion that one is directly manipulating the objects of concern.
>
> The system should be responsive, with no delays between execution and the results, except where those delays are appropriate for the knowledge domain itself.
>
> The interface should be unobtrusive, not interfering or intruding. If the interface itself is noticed, then it stands in a third-person relationship to the objects of interest, and detracts from the directness of the engagement.

In order to have a feeling of direct engagement, the interface must provide the user with a world in which to interact. The objects of that world must feel like they are the objects of interest, that one is doing things with them and watching how they react. In order for this to be the case, the output language must present representations of objects in forms that behave in the way that the user thinks of the objects behaving. Whatever changes are caused in the objects by the set of operations must be depicted in the representation of the objects. This use of the same object as both an input and output entity is essential to providing objects that behave as if they are the real thing. It is because an input expression can contain a previous output expression that the user feels the output expression is the thing itself and that the operation is applied directly to the thing itself.

In addition, all of the discussions of semantic and articulatory directness apply here too, because the designer of the interface must be concerned with what is to be done and how one articulates that in the languages of interaction. But the designer must also be concerned with creating and supporting an illusion. The specification of what needs to be done and evidence that it has been done must not violate the illusion, else the feeling of direct engagement will be lost.

One factor that seems especially relevant to maintaining this illusion is the form and speed of feedback. Rapid feedback in terms of changes in the behavior of objects not only allows for the modification of actions even as they are being executed, but also supports the feeling of acting directly on the objects

themselves. It removes the perception of the computer as an intermediary by providing continual representation of system state. In addition, rapidity of feedback and continual representation of state allows one to make use of perceptual faculties in evaluating the outcome of actions. We can watch the actions take place, monitoring them much like we monitor our interactions with the physical world. The reduction in the cognitive load of mentally maintaining relevant information and the form of the interaction contribute to the feeling of engagement.

5. A SPACE OF INTERFACES

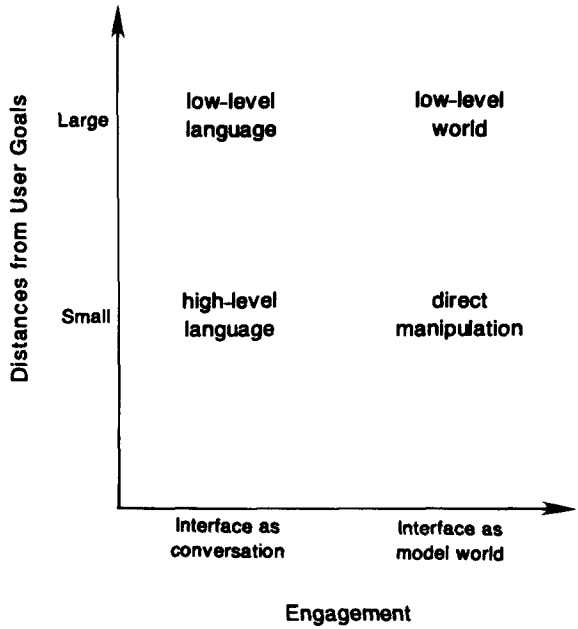
Distance and engagement are depicted in Figure 7 as two major dimensions in a space of interface designs. The dimension of engagement has two landmark values: One is the metaphor of interface as conversation; the other is the metaphor of interface as model world. The dimension of distance actually contains two distances to be spanned: semantic and articulatory distances, the two kinds of gulfs that lie between the user's conception of the task and the interface language.

The least direct interface is often one that provides a low-level language interface, for this is apt to provide the weakest semantic match between intentions and the language of the interface. In this case, the interface is an intermediary between the user and the task. Even worse, it is an intermediary that does not understand actions at the level of description in which the user likes to think of them. Here the user must translate intentions into complex or lengthy expressions in the language that the interface intermediary can understand.

A more direct situation arises when the central metaphor of the interface is a world. Then the user can be directly engaged with the objects in a world; but still, if the actions in that world do not match those that the user wishes to perform within the task domain, getting the task done may be a difficult process. The user may believe that things are getting done and may even experience a sense of engagement with the world, yet still be doing things at too low a level. This is the state of some of the recently introduced direct manipulation systems: They produce an immediate sense of engagement, but as the user develops experience with the system, the interface appears clumsy, to interfere too much, and to demand too many actions and decisions at the wrong level of specification. These interfaces appear on the surface to be direct manipulation interfaces, but they fail to produce the proper feelings of direct engagement with the task world.

Closing the distance between the user's intentions and the level of specification of the interface language allows the user to make efficient specifications of intentions. Where this is done with a high-level language, quite efficient interfaces can be designed. This is the situation in most modern integrated pro-

Figure 7. **A space of interfaces. The dimensions of distance from user goals and degree of engagement form a space of interfaces within which we can locate some familiar types of interfaces. Direct manipulation interfaces are those that minimize the distances and maximize engagement. As always, the distance between user intentions and the interface language depends on the nature of the task the user is performing.**



gramming environments. For some classes of tasks, such interfaces may be superior to direct manipulation interfaces.

Finally, the most direct of the interfaces will lie where engagement is maximized, where just the right semantic and articulatory matches are provided, and where all distances are minimized.

6. PROBLEMS WITH DIRECT MANIPULATION

Direct manipulation systems have both virtues and vices. For instance, the immediacy of feedback and the natural translation of intentions to actions make some tasks easy. The matching of levels of thought to the interface language—semantic directness—increases the ease and power of performing some activities at a potential cost of generality and flexibility. But not all things should be done directly. For example, a repetitive operation is probably best done via a script, that is, through a symbolic description of the tasks that

are to be accomplished. Direct manipulation interfaces have difficulty handling variables, or distinguishing the depiction of an individual element from a representation of a set or class of elements. Direct manipulation interfaces have problems with accuracy, for the notion of mimetic action puts the responsibility on the user to control actions with precision, a responsibility that is sometimes best handled through the intelligence of the system and sometimes best communicated symbolically.

A more fundamental problem with direct manipulation interfaces arises from the fact that much of the appeal and power of this form of interface comes from its ability to directly support the way we normally think about a domain. A direct manipulation interface amplifies our knowledge of the domain and allows us to think in the familiar terms of the application domain rather than in those of the medium of computation. But if we restrict ourselves to only building an interface that allows us to do things we can already do and to think in ways we already think, we will miss the most exciting potential of new technology: to provide new ways to think of and to interact with a domain. Providing these new ways and creating conditions that will make them feel direct and natural is an important challenge to the interface designer.

Direct manipulation interfaces are not a panacea. Although with sufficient practice by the user many interfaces can come to feel direct, a properly designed interface, one which exploits semantic and articulatory directness, should decrease the amount of learning required and provide a natural mapping to the task. But interface design is subject to many tradeoffs. There are surely instances when one might wisely trade off directness for generality, or for more facile ways of saying abstract things. The articulatory directness involved in pointing at objects might need to be traded off against the difficulties of moving the hands between input devices or of problems in pointing with great precision.

It is important not to equate directness with ease of use. Indeed, if the interface is really invisible, then the difficulties within the task domain get transferred directly into difficulties for the user. Suppose the user struggles to formulate an intention because of lack of knowledge of the task domain. The user may complain that the system is difficult to use. But the difficulty is in the task domain, not in the interface language. Direct manipulation interfaces do not pretend to assist in overcoming problems that result from poor understanding of the task domain.

What about the claims for direct manipulation? We believe that direct manipulation systems carry gains in ease of learning and ease of use. If the mapping is done correctly, then both the form and the meaning of commands should be easier to acquire and retain. Interpretation of the output should be immediate and straightforward. If the interface is a model of the task domain, then one could have the feeling of directly engaging the problem of interest itself. It is sometimes said that in such situations the interface disappears. It is

probably more revealing to say that the interface is no longer recognized as an interface.

But are these desirable features? Are the trade-offs too costly? As always, we are sure that the answer will depend on the tasks to be accomplished. Certain kinds of abstraction that are easy to deal with in language seem difficult in a concrete model of a task domain. When we give up the conversation metaphor, we also give up dealing in descriptions, and in some contexts, there is great power in descriptions. As an interface to a programming task, direct manipulation interfaces are problematic. We know of no really useful direct manipulation programming environments. Issues such as controlling the scope of variable bindings promise to be quite tricky in the direct manipulation environments. Will direct manipulation systems live up to their promise? Yes and no. Basically, the systems will be good and powerful for some purposes, poor and weak for others. In the end, many things done today will be replaced by direct manipulation systems. But we will still have conventional programming languages.

On the surface, the fundamental idea of a direct manipulation interface to a task flies in the face of two thousand years of development of abstract formalisms as a means of understanding and controlling the world. Until very recently, the use of computers has been an activity squarely in that tradition. So the exterior of direct manipulation, providing as it does for the direct control of a specific task world, seems somehow atavistic, a return to concrete thinking. On the inside, of course, the implementation of direct manipulation systems is yet another step in that long, formal tradition. The illusion of the absolutely manipulable concrete world is made possible by the technology of abstraction.

Acknowledgments. We thank Ben Shneiderman for his helpful comments on an earlier draft of the chapter, Eileen Conway for her aid with the illustrations, and Julie Norman and Sondra Buffett for extensive editorial comments.

Support. The research reported here was conducted under Contract N00014-85-C-0133, NR 667-541 with the Personnel and Training Research Programs of the Office of Naval Research and with the support of the Navy Personnel Research and Development Center. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the sponsoring agency.

REFERENCES

Black, J. B., & Moran, T. P. (1982). Learning and remembering command names. *Proceedings of the Human Factors in Computer Systems Conference,* 8-11. New York: ACM.

Black, J. B., & Sebrechts, M. M. (1981). Facilitating human-computer communication. *Applied Psycholinguistics, 2,* 149-177.

Borning, A. (1979). *ThingLab: A constraint-oriented simulation laboratory* (Tech. Rep. No. SSL-79-3). Palo Alto, CA: Xerox Palo Alto Research Center.

Budge, B. (1983). *Pinball construction set* [Computer program]. San Mateo, CA: Electronic Arts.

Buxton, W. (1986). There's more to interaction than meets the eye: Some issues in manual input. In D. A. Norman & S. W. Draper (Eds.), *User centered system design: New perspectives on human-computer interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Carrol, J. M. (1985). *What's in a name? An essay in the psychology of reference.* New York: Freeman.

diSessa, A. A. (1985). A principles design for an integrated computational environment. *Human-Computer Interaction, 1,* 1-47.

Draper, S. W. (1986). Display managers as the basis for user-machine communication. In D. A. Norman & S. W. Draper (Eds.), *User centered system design: New perspectives on human-computer interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Galton, F. (1894). Arithmetic by smell. *Psychological Review, 1,* 61-62.

Hollan, J. D., Hutchins, E., & Weitzman, L. (1984). Steamer: An interactive inspectable simulation-based training system. *AI Magazine, 5,* 15-27.

Hollan, J. D., Stevens, A., & Williams, M. D. (1980). Steamer: An advanced computer-assisted instruction system for propulsion engineering. *Proceedings of Summer Computer Simulation Conference,* 400-404. Arlington, VA: AFIPS Press.

Kay, A. (1984, September). Computer software. *Scientific American,* 52-59.

Laurel, B. K. (1986). Interface as mimesis. In D. A. Norman & S. W. Draper (Eds.), *User centered system design: New perspectives on human-computer interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Minksy, M. R. (1984, July). Manipulating simulated objects with real-world gestures using a force and position sensitive screen. *Computer Graphics,* 195-203.

Norman, D. A., & Draper, S. W. (Eds.). (1986). *User centered system design: New perspectives on human-computer interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Perlis, A. J. (1982). Epigrams on programming. *SIGPLAN Notices, 17*(9), 7-13.

Shneiderman, B. (1974). A computer graphics system for polynomials. *The Mathematics Teacher, 67*(2), 111-113.

Shneiderman, B. (1982). The future of interactive systems and the emergence of direct manipulation. *Behavior and Information Technology, 1,* 237-256.

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

Sutherland, I. E. (1963). Sketchpad: A man-machine graphical communication system. *Proceedings of the Spring Joint Computer Conference,* 329-346. Baltimore, MD: Spartan Books.

HCI Editorial Record. This is an invited paper based on a draft of April 1, 1985. Final manuscript received October 3, 1985. —*Editor*