(In S. R. Fussell & R. J. Kreuz (Eds.), *Social and cognitive psychological approaches to interpersonal communication (pp. 201-225)*. Hillsdale, NJ: Lawrence Erlbaum. Copyright © 1998 by Lawrence Erlbaum. All rights reserved.)

9

The Grounding Problem in Conversations With and Through Computers

Susan E. Brennan
State University of New York, Stony Brook

In this chapter I look at human–computer interaction as a kind of coordinated action that bears many similarities to conversational interaction. In human–computer interaction, a computer can be both a medium to communicate *through* and a partner to communicate *with*. I consider how people coordinate their activities with other people electronically, over time and distance, as well as how they communicate with computers as interactive partners, regardless of whether the currency of interaction is icons, text, or speech. The problem is that electronic contexts are often impoverished ones. Many of the errors that occur in human–computer interaction can be explained as failures of *grounding*, in which users and systems lack enough evidence to coordinate their distinct knowledge states. Understanding the grounding process provides not only a systematic framework for interface designers who want to understand and improve human–computer interaction, but also a testbed for cognitive and social psychologists who seek to model the effects of different contexts and media upon language use.

Conversations *Through* Computers

To communicate successfully, two people need to coordinate not only the content of what they say, but also the process of saying it. Consider Don, sitting in his office early one morning, typing an email message to Michael, whose office is in another building. If Don wants to get Michael to join him for lunch at a particular restaurant, he cannot simply write, “Let’s meet at Arizona at 1:00.” There are many points at which something could go wrong. Don needs to be confident that Michael is able to receive the message (is his computer on?), is attentive enough to know there is a message (or is he playing Tetris again?), has received the message (or is his mail server down?), knows that the message is from Don (and not someone else), can figure out what Don means (Arizona is that restaurant with the great desserts on Manhattan's Upper East Side), and is willing and able to commit himself to the action it proposes (and does not have an impending deadline or early afternoon meeting). So after sending his invitation, Don awaits evidence that Michael has received, understood, and committed to the invitation. Meanwhile, Michael does not begin hunting for a cab as soon as he gets Don’s message, but sends an email reply. If their electronic connection is unreliable, or if Michael needs to further clarify or modify their plans, they may exchange still more email before they consider their plan to meet at the restaurant to be common ground. Depending on time and other pressures, Don may opt to telephone Michael if an email response is not forthcoming. In this way, Don and Michael engage in the process of *grounding* in order to come to the mutual belief that they understand one another sufficiently well for the purpose at hand.

The grounding process has been described within a framework that views communication as a form of collaborative action (Brennan, 1990a; Clark, 1996; Clark & Brennan, 1991; Clark & Schaefer, 1989; Clark & Wilkes-Gibbs, 1986; Isaacs & Clark, 1987; Schober & Clark, 1989). According to this view, for a speaker (take Don, in this example) to contribute to a conversation, it is not sufficient for him simply to produce an utterance. He must also acquire sufficient evidence that the utterance has been heard and understood as intended. But how he grounds the utterance will vary, depending on several factors. One kind of factor involves Don's current purposes; if he really hates being stood up in public places, then he will require strong evidence that Michael is coming before concluding that the two of them have a lunch appointment. On the other hand, if Don will be hanging out at the restaurant bar anyway and it is not so important that Michael show up on time, then he will require less evidence. Depending on their purposes, a speaker and an addressee adjust their *grounding criteria* to seek and provide more or less evidence that an utterance presented by the speaker has been accepted by the addressee (Clark & Schaefer, 1989; Clark & Wilkes-Gibbs, 1986; Wilkes-Gibbs, 1986).

Another factor that affects how grounding takes place is the communication medium itself. Depending on whether the medium is face to face, telephone, email, text teleconferencing, video teleconferencing, fax, or postal mail, different constraints are placed on the exchange of evidence (Brennan, 1990a; Clark & Brennan, 1991). For instance, the immediacy with which two people can exchange evidence is critical. If Don and Michael are able to produce and receive a rapid succession of turns—for instance, if they are using an interactive electronic "chat" program where they can simultaneously type and see what the other is typing, or even better, if they are talking on the phone, or best of all, if they are talking face to face—then it is much faster and easier for them to reach the mutual belief that they understand one another than if they are sending email messages or faxes or even worse, postal mail. This is true because producing an utterance, knowing whether an addressee has attended to it, and turning over the conversational floor to that addressee for a response cost relatively less in time and effort in a medium in which two people can be temporally co-present than in one in which they cannot. In media where people are not co-present (in the same place, at the same time) and utterances are not ephemeral, such as with email, faxes, and postal letters, people tend to ground larger installments than in spoken conversation. In these ways, the affordances of a medium impose particular costs on the grounding process and on how grounding shapes the conversations conducted over that medium (Clark & Brennan, 1991).

Many studies have described how the form of communication differs across media (Cohen, 1984; Ochsman & Chapanis, 1974). Grounding provides a useful framework with which to predict and explain these differences (Clark & Brennan, 1991; Whittaker, Brennan, & Clark, 1991).

Conversations *With* Computers

The grounding process is useful as well in understanding what happens when the interactive partner is a computer. Consider what happens when Don returns from lunch and logs in to his computer. He means to copy some files into a public directory so that his supervisor can review them. He types, “copy report.97 public.” The system returns a prompt. Then he copies another file by typing, “copy budget.97 public.” Again, a prompt. Later, he is surprised to discover that *public* does not contain his two intended files after all. It turns out that he had forgotten to create a *directory* called “public” before trying to copy his files, and instead he wrote the files, one after another, into a *file* named “public,” the second file overwriting the first. Many DOS and UNIX®[1] users have experienced this kind of mishap. They soon learn to check to see whether their commands have had the desired effect; for instance, after copying, moving, or deleting a file, they may list the contents of a directory to discover whether all is as expected. Such checking behavior is a way of grounding with an uncooperative operating system.

Seeking evidence that things are on track is not unique to situations that involve communication. Many other sorts of activities require people to express their intentions as action sequences and then to evaluate the results of their actions against their intentions (Hutchins, Hollan, & Norman, 1986; Norman, 1990). Experience with the physical properties of the world, with cause–and–effect sequences, and with perceptual feedback can make this process fairly straightforward for adults dealing with physical objects. Many objects have obvious affordances that enable people to recognize what they are for and how to use them (Norman, 1990; see also Gibson, 1977). In the physical world, actions often result in incremental perceptual feedback that people can use to evaluate their progress toward a goal. But this is not always the case in an electronic world; affordances and the results of actions are often not represented explicitly.

Human conversation and human–computer interaction are both coordinated activities. In both, people need to be able to seek evidence that they have been understood and to provide evidence about their own intentions. However, unless a system’s designers have been attentive to the system’s user interface, or unless the user is an expert, the evidence needed for grounding can be very difficult to get, and errors can be very difficult to recognize. It often falls to users to put in the extra effort needed to try to keep things on track. This is what I call *the grounding problem in human–computer interaction*. This

[1] UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

problem exists in conversations both with and through computers—that is, whether the computer is primarily an object to interact with (in the case of single-user applications like word processors, database query programs, spreadsheets, or autonomous software agents) or a medium for interacting with other people (in the case of email or teleconferencing programs). Next, I present some background about how computers evolved into interactive partners. Then I consider how the grounding problem has been addressed or ignored in different kinds of human–computer dialogs, both graphical and language-based. Finally, I discuss how electronic interfaces could better support users' grounding activities with spoken dialog systems.

THE EVOLUTION OF HUMAN–COMPUTER INTERACTION

To consider how support has evolved for enabling people to ground their activities with computers, it is necessary to consider how computers evolved to be interactive in the first place.[2] In the early days of computing, people did not "interact" with computers. In the 1940s, engineers instructed Eniac, the first general-purpose electronic computer, by rewiring it. By the 1950s, programmers were able to control Univac with typewritten commands rather than with dials and switches. Batch processing was yet another advance, one that made computers available to many more users. Users scripted out every instruction in advance and represented the instructions by punching tiny square holes in precise locations on cardboard cards. Then they carefully transported their huge stacks of cards to the local computing center, waited a day or so, and then returned with the hope that none of their cards had been mispunched or misordered. As often as not, one card would have been out of order or mispunched, and then users were faced with the nightmarish task of trying to figure out what was wrong (or simply starting over). The prevailing metaphor for a computer was a giant calculator, not an interactive partner.

By the late 1950s, PDP-1 computers became available; these so-called *minicomputers* were the size of only two refrigerators. A PDP-1 could be controlled by a high-speed paper tape as well as by punched cards, and the tapes could be changed while the machine was running. So, in a crude fashion, the computer operator could interact with the computer. An experimental psychologist named J. C. R. Licklider envisioned future possibilities for interacting with such machines (Licklider, 1960, as quoted in Rheingold, 1985, p. 40):

> The equipment will answer questions. It will simulate the mechanisms and models, carry out the procedures, and display the results to the operator. It will transform data, plot graphs (. . . in whatever way the

[2] For a comprehensive and entertaining history, see Howard Rheingold's (1985) book, *Tools for thought: The people and ideas behind the next computer revolution*. This is that rare account with which the computer scientists described in its pages actually concur.

human operator specified, or in several alternative ways if the human operator is not sure what he wants).

During the 1950s, an electrical engineer named Doug Engelbart suggested that keyboards could be hooked up to computers, users could be taught to type, and computers could display information on cathode-ray screens. This was considered a crazy idea by his colleagues in academia and industry, but he eventually received a small amount of funding from the Air Force to pursue it. He also proposed that computers could be used for text editing and, in the mid 1960s, constructed the first mouse input device (Rheingold, 1985). These ideas were important in enabling the kind of human–computer interaction we take for granted today.

The First Conversational Computer. In 1963, Ivan Sutherland presented the world with what is widely acknowledged to be the first human–computer interface to support real-time graphical human–computer interaction. His *Sketchpad* system included a display, tablet, stylus, and graphical elements that behaved like physical objects and enabled a user and a system "to converse rapidly through the medium of line drawing" (Sutherland, 1963). Users conversed with Sketchpad by pointing and drawing. The system responded by updating the drawing immediately, so that the relationship between the user's action and the graphical result was clear. Interestingly enough, Sketchpad launched not one but two of the most influential metaphors that have been developed for computing. First, even though Sketchpad was primarily graphical, Sutherland saw it as *conversational*. That is, it was highly responsive; feedback was so timely and relevant that it could be considered analogous to backchannels in human conversation (these include timely responses such as acknowledgments, nods, and eye contact; see Yngve, 1970). The second metaphor grew out of the feeling users had of directly manipulating graphical objects. Sketchpad launched a style of human–computer interaction that has been labeled *direct manipulation* (Shneiderman, 1982, 1983) and with it, the gradual but profound revolution in the design and usability of computers that has led to today's desktop and windows-based systems.

Conversation versus Direct Manipulation

From the late 1970s until the early 1980s (and much later for some), users typically interacted with computer systems through a teletype (TTY) interface. The user would type a command and then receive a text response printed on the screen, along with a prompt signaling that the system was ready for the next command. Some applications, as well as operating systems such as VMS, DOS, and UNIX, still offer this type of interface. Such command-style interfaces are often categorized together with interfaces that rely on a *natural language* such as English (e.g., Schneiderman, 1992). Such interfaces are primarily language-based as opposed to graphical; they linearize human–computer interaction as a series of alternating turns; and they sometimes act as an agent or intermediary between users and their goals (see, e.g., Brennan, 1990b; Laurel, 1990). Sometimes these so-called "conversational" interfaces are contrasted with direct manipulation interfaces, as if they represent two distinct styles of interaction (Sutherland's early insights notwithstanding). For instance, Shneiderman (1992, 1993) has claimed that direct manipulation interfaces are superior to language-based interfaces. But I argue here that the features of direct manipulation interfaces that make them so usable are the very features that work so well in human conversation.

Direct manipulation interfaces provide excellent support for grounding. The objects of interest are continuously represented, actions are incremental, reversible, and physical (rather than syntactically complex), and there is immediate visual feedback about the effects of actions (Hutchins et al., 1986; Schneiderman, 1982). These interfaces are relatively easy for novices to learn to use and for occasional users to remember. The prevailing metaphor for computing is as a set of tools and objects with predictable uses and characteristics.

In contrast, with a language-based interface, the dominant metaphor is that of a more or less intelligent *agent*, a process to whom the user can delegate actions. Unfortunately, command language and query language interfaces can be difficult for novices and occasional users because they are underdetermined; that is, they require users to remember a set of commands and precise syntax, know exactly how to refer to the information and objects in a domain, and keep track of the current context. Schneiderman (1992), who classifies natural language and speech interfaces in the same general category as command and query language interfaces, points out that with natural language, users aren't burdened with learning and remembering special syntax; but for him, this advantage does not outweigh disadvantages such as unpredictability. Schneiderman (1986) argues:

> . . . human–human interaction is not necessarily an appropriate model for human operation of computers. Since computers can display information 1,000 times faster than people can enter commands, it seems advantageous to use the computer to display large amounts of information and allow novice and intermittent users simply to choose among the items.

The problems with direct manipulation dialogs include the possibility for users to become overloaded, as well as the difficulty of visibly representing complex queries and relationships among objects. To the extent that software is predictable and consistent, a tool metaphor works well; but as soon as the tool is expected to do any sophisticated processing, or if it breaks down or needs more information, or if what is going on cannot be visually represented, then it is more like an agent, with whom people must coordinate and communicate. The advantages of language-based interfaces over direct manipulation typically include the ability to use negation and quantification in queries, distinguish individuals from kinds, search very large databases, issue commands over sets, filter and request information in novel ways, and perform actions that are not in the here and now. Spoken input, although it can lead to recognition errors, has additional advantages: It does not rely on the user's hands or eyes, and it can work at a distance, over the phone, and in the dark.

Although speech and natural language technology has advanced considerably over the past couple of decades, the potential that many researchers have foreseen for these technologies in the human–computer interface has yet to be realized. Relatively few users depend on speech and natural language to interact with their computers. This is because simply having language as the currency of interaction is not enough to make an interface conversational. There are virtually no speech or natural language systems where as much theory and systematic effort has been devoted to the design of the dialog model as to the underlying signal processing or syntactic parsing mechanisms.

The relative success of direct manipulation interfaces, I believe, is not due to their literal resemblance to tools or to any particular "naturalness"; even a tool must be mastered, and graphical representations can be just as arbitrary or convention-based as linguistic ones. I argue that it is not whether the metaphor is a tool or an agent that makes or breaks an interface, but the extent to which the interface supports the grounding process. It just so happens that the architecture of a direct manipulation interface is more likely to support the grounding process than is the sometimes ad hoc feedback provided in a language-based system.

GROUNDING IN HUMAN–COMPUTER INTERACTION

Conversations with other people are rarely fluent and without mishap, and people do not expect them to be. Yet some human–computer interfaces are designed as if people rarely make errors (Lewis & Norman, 1986). In this section I discuss some of the ways in which computer interfaces help or hinder users in grounding their actions by providing feedback, representing shared context, enabling referring, and supporting incremental actions. Most of these examples are drawn from the experiences of college students trying to use popular applications and systems.

Feedback

One major problem with older command line interfaces to operating systems like VMS, DOS, and UNIX is that they often fail to provide appropriate feedback, apart from a bare prompt when ready for input. To discover whether a command has had a particular effect, users may need to search for evidence, as Don had to when he copied his UNIX files. Information about the status of an application is a powerful kind of evidence with which users can ground their activities. Sometimes this evidence is fortuitous, such as the whirring of a hard drive, a symptom that it is working. Sometimes status information is provided more explicitly by a software or hardware designer, such as the lights on a disk drive that borrow convention from traffic lights and flash green, yellow, or red. In the early 1980s, twinkling *run bars* in the bottom margin of windows containing interpreted Lisp code provided reassurance that a user's programs had not yet crashed, and command line systems sometimes produced a row of elliptical dots, showing that a command was in the process of being executed. Such features were strikingly innovative back then and are now commonplace. Many applications now provide static status information in the form of cursors that change to clocks or hourglasses or other symbols of waiting, or dynamic status information (time-elapsed or time-remaining) in the form of rectangles that fill slowly as a file is loaded or converted or saved.

Although these status messages do not exactly mimic what people do in conversation (imagine asking a hard question of someone who responded by turning over an hourglass!), they mimic human feedback behavior in spirit. Not only do people in conversation provide backchannel responses (Yngve, 1970), but they have other ways of signaling their metacognitive states in conversation, such as by filling a pause before an answer with "um" or "uh" (Brennan & Williams, 1995; Smith & Clark, 1993). If a speaker were to hesitate silently before answering a question, this pause would license unwanted implicatures; fillers such as *um* and *uh* display the fact that the speaker is working on producing an utterance. Hearers can use this information (the presence or absence of fillers) to make accurate inferences about the speaker's commitment to an answer based on the display that precedes the answer (Brennan & Williams, 1995). An unexpected delay licenses the inference that a conversational partner is having difficulty. Consider this example from a conversation between two people who were discussing objects in a laboratory experiment by Brennan and Clark (1996):

D: number 3 is a, a car.
(pause)
did you get that?
M: yes

After the pause, *D* explicitly elicited evidence from *M*. Typically in a task like this, *M* will acknowledge *D*'s utterance as soon as possible with "mm hm" or "okay." An immediate acknowledgment saves a conversational turn; when *D* expects an acknowledgment and does not get one, she will probably infer that *M* has a problem. If *M* puts in the minimal effort required to provide a timely acknowledgment now, then *M* and *D* won't need to put in more effort to repair the problem later. In this way, both *M* and *D* share the responsibility for getting utterances understood, and this process is governed by a principle that Clark and Wilkes-Gibbs have called *least collaborative effort* (1986).

Although status indicators provide feedback that saves users the effort of hunting for evidence about what the system is up to, many interface functions are not designed with least collaborative effort in mind. As with human partners, the timing of feedback is critical in grounding with a computer, although the consequences of delayed feedback differ from context to context. If a system is slow to provide feedback, users may assume their inputs have not been received and may continue to click the mouse button or hit the carriage return. This practice can queue up unintended inputs that cause the wrong window to be selected or the wrong file to be opened. Or, in the absence of any expectations, users may simply wait, as did one novice who typed a query to a library database program. After 5 minutes with no feedback, he asked an expert, who reminded him to hit the carriage return.

Because of the obvious asymmetries in the capabilities of human and computer partners, most of the responsibility for coordinating joint activities with systems and for minimizing effort falls on users. At the same time, systems that provide feedback only about how commands have failed rather than about how they have been processed do not give users enough evidence to ground their actions. In these cases, users are forced to search for positive evidence, not only that their commands were executed, but that they were executed as intended. Mishaps such as copying several files over a single file (instead of moving them all to a directory) would be avoidable if the system's response to the user's move command were to include the name of the target file as positive evidence of how the command was interpreted or if potentially destructive commands were to require confirmation. For instance, in recent versions of DOS, as Don tried to copy his second file, he would have received a request for confirmation such as "Overwrite file: public?" This message contains at least two clues that something unexpected is about to happen: *Overwrite* describes a destructive action, and *file: public* tells Don that what he thought was a directory is really a file. If Don is paying attention (and does not just respond with an automatic "yes"), then he can compare the evidence with his goal and avoid the error.[3] Note that the system would not recognize any problem with this sequence of commands, because they are perfectly legal; it is up to the user to identify the error. For this to happen, the user needs to have not only evidence about system errors, but also evidence about successful actions.

Shared Context and Referring

In conversation, people construct and maintain discourse models that represent the entities under discussion as well as the relevant relationships between these entities. As two people in a conversation accumulate *common ground*, they presumably construct discourse models more similar to each other's, containing information that both believe to be shared. They rely on their common ground in referring to objects (Clark & Marshall, 1981); referring

[3] This kind of feature has been called a forcing function (see Norman, 1990). Forcing functions are safety nets that are provided wherever people meet technology (e.g., not only in human-computer interfaces but also in appliances, architecture, automobiles, etc.); they make it difficult or impossible to commit an error, and to that extent, they are enormously useful in enabling people to ground their actions with objects. Forcing functions have two major limitations: they typically prevent common errors that have been foreseen in advance, and they can sometimes be unnecessary and therefore annoying, which causes people to find ways to defeat them (such as ignoring a warning message or responding automatically to a request for confirmation).

expressions become more efficient over repeated use (Krauss & Weinheimer, 1966) and reflect jointly achieved perspectives that may not be understood by third parties who have not participated in the conversations (Clark & Wilkes-Gibbs, 1986; Schober & Clark, 1989; Wilkes-Gibbs & Clark, 1992). During the process of grounding, people exchange evidence until they reach the mutual belief that they are talking about the same thing (Brennan, 1990a; Clark & Schaefer, 1987, 1989). Even people with very different language abilities (such as native and non-native speakers of English) rely on the process of grounding (Bortfeld & Brennan, 1997).

Because many interfaces do not maintain and use explicit discourse models, problems arise when users transfer their expectations about shared context from their conversations with other people to their interactions with computers. People expect natural language interfaces to be able to handle ellipsis and pronominal references to previously mentioned entities, just as they expect human partners to do these things (Brennan, 1991). So errors occur when natural language interfaces treat each query in a dialog as an independent event. Another source of problems is that applications are often not presented as coherent interactive partners, with respect to their output messages. Consider this example in which an undergraduate tried to use the command interface to a library database:

System: PLEASE SELECT COMMAND MODE
User: >Please find an author named Octavia Butler.
System: INVALID FOLIO COMMAND: PLEASE

Here, gratuitous user friendliness in the form of the word *please* led this user astray. In a study by Brennan & Ohaeri (1994), users of natural language interfaces tended to use more indirect language with computer partners that used indirect language. This was probably less a matter of saving face (see Brown & Levinson, 1987; Holtgraves, 1997) than simply adopting the kinds of phrasings that the partner used. Whether people are communicating with other people or with computer partners, and whether they use speech or text, they tend to converge with their partners in the terms they use (Brennan, 1996, in press). By using the same expression, two partners in conversation mark the mutual belief that they are referring to the same object (Brennan & Clark, 1996). As in the library database example, some speech- and language-based interfaces do not behave this way; their output messages are inconsistent with the input they can handle (Brennan, 1988, 1991).

Most direct manipulation interfaces do a better job of representing and sharing context[4] than do most language-based interfaces, because the objects of interest (such as icons) are continuously represented on the screen (Hutchins et

[4] By *shared context*, I mean something akin to common ground. However, in order to avoid anthropomorphizing computer systems and to acknowledge the basic asymmetry between the partners in any human-computer dialog, I will reserve the term *common ground* for interactions between human beings, who have the capability to be mutually aware that they share knowledge.

al., 1986). What a direct manipulation interface displays on the screen amounts to the current state of its discourse model (Brennan, 1990b); what is on the screen is also a good estimate of the context it shares with a user. What is highlighted at any moment denotes what the system estimates to be the shared focus of attention. Of course, the user may be attending to something off the screen; some research prototypes have actually attempted to explicitly establish co-presence by modeling the user's position and orientation in an office (e.g., Schmandt, Arons, & Simmons, 1985).

Referring is relatively straightforward in a direct manipulation interface because discourse entities are embodied in graphical icons and the same icons can serve as inputs to and outputs from commands (Draper, 1986). For instance, to ask about what files are contained in a folder, a user of the Macintosh®[5] operating system need only double-click on the folder and the folder will flip open to display its files. Then the user can refer to any visible file by clicking on it; one additional gesture, such as another click or dragging, initiates a command with respect to that object. Cursors change location to represent attentional shifts on the screen; they change shape to represent larger context (or *mode*) shifts (Reichman, 1986). Transitions between system states can be expressed explicitly by animation; for example, when a folder is opened, its icon changes to show a transition to its open state.

Not all so-called direct manipulation interfaces enable grounding equally well. On Macintosh-style interfaces (including Microsoft's Windows 95®[6]), objects of the same types are represented as icons that look alike, and applications look different from documents. There is a consistent spatial metaphor for referring; objects of all sorts can be moved around by the user, and they stay put when they are moved. This is not the case in Microsoft's Windows 3.1® and its predecessors, often considered a poor excuse for a direct manipulation interface. In Windows 3.1, applications are typically represented by identical icons (only the labels are different), documents are not even visible, and although objects can be moved, they return to their old locations when the system is rebooted (unless the user evokes an additional command), a violation of the spatial metaphor.

Although the graphical feedback used in some desktop and window-style interfaces to represent and update shared context works well for users with a bit of experience, it may not work for novices. Even the Macintosh desktop, the gold standard of direct manipulation interfaces, depends on feedback conventions that must be learned. For instance, on the Macintosh desktop (and its imitators), the window that receives input is typically the one that appears to be on top of the others; it is also highlighted with darker borders. If users are unaware of this convention, their input may go to an unintended window. UNIX window managers, on the other hand, are often customizable and can therefore rely on different conventions; some do not use highlighting to show which window is the current one, and some enable typing wherever the

[5] Macintosh is a registered trademark of Apple Computer, Inc.

[6] Windows 3.1 and Windows 95 are registered trademarks of Microsoft Corporation.

mouse cursor appears, even on windows that are partly occluded. And in direct manipulation interfaces, referring to objects can become complicated by *aliases*, pointers to objects (such as applications or folders) that resemble the actual object, but do not always behave the same. On the Macintosh, an alias has an italicized label and looks slightly different from a regular icon. Aliases are convenient for those in the know, but as conventions that depart from the spatial metaphor, they can be confusing.[7] Once learned, these conventions seem so natural that experts find it inconceivable that they could be opaque to novices.

Errors can result from combinations of inadequate feedback and failure to model context. For example, a common problem for novices is when newly saved files appear to be lost. In most Macintosh applications, a dialog window appears when "Save" is selected for a new file and the user is prompted to name the file by a cursor that appears in a slot labeled "Save Current Document as." However, above this slot is a scrolling window and pull-down menu that displays the folder in which the file will be saved, an important piece of information. Sometimes this folder coincides with what the user would consider to be the current context, and so there is no problem; the file gets saved right where the user expects it to be. But often, the default location happens to be the one where the application program is located or where files were saved the last time the application was used. This situation leads to confusion when the user is unaware of this convention or fails to notice where the file is going. The problem is that the destination is not labeled as such—it is labeled with only the name of the default current folder, next to a small triangle pointing down. The triangle is a convention indicating that the label is really a pull-down menu. To novices, who may not even notice the tiny triangle, nothing in this representation says that (a) the label shows the current context, according to the machine, and (b) this context can be reset by pulling down the menu and stepping up the hierarchy of folders in order to reach the desired folder. Novices, for whom this error is common, panic when they cannot find their files where they thought they left them. Experienced users make this error too, especially if they are in a hurry and ignore the default folder; for them it is only a minor annoyance (but an annoyance nevertheless) to hunt down the file using the "Find" command and move it to the right location. This error could be prevented if the destination were saliently labeled as such or if saving a file required confirmation of the destination, especially the first time the application is opened, when the context is especially likely to be ambiguous.

Grounding Incrementally

The grounding process requires that partners be able to seek incremental evidence of each other's understanding, as well as provide such incremental evidence about their own understanding. Such evidence may be either positive

[7] I thank Susan Fussell for bringing this example to my attention.

(evidence that a speaker believes everything is on track) or negative (evidence that a speaker has detected a problem). In the next example, speaker *B* arrives at an interpretation of speaker *A*'s first utterance that *A* did not intend:

A: You don't have any nails, do you?
B: *<pause>*
Fingernails?
A: No, nails to nail into the wall.
<pause>
When I get bored here I'm going to go put up those pictures.
B: No.

Clarification subdialogs (see Jefferson, 1972) enable people in conversation to coordinate their individual knowledge states. In this example, before *B* understands and answers *A*'s question, she initiates a clarification subdialog that nests within the question and its answer. Although clarification subdialogs are relatively simple to manage in human conversation, this is often not the case with computers. Consider a user who evokes on-line help in the middle of trying to accomplish some task. Many help systems are not context-sensitive; they begin by presenting a long index of available topics, regardless of what the user and application are in the middle of doing. This would be like *A* responding to *B*'s request for clarification by ignoring its content and simply providing a long list of all the terms that *A* knows for *B* to select from.

In human conversation, grounding is frequently done in small increments, as when speakers give a telephone number three or four digits at a time, waiting for acknowledgments from addressees after each installment (Clark & Schaefer, 1987). Speakers trade off the costs of grounding with the benefits; exchanging evidence of understanding is harder in some communication media than in others, and this fact affects the strategies and techniques people choose for grounding (Clark & Brennan, 1991). Speakers are more likely to break a contribution up into installments when they have a high grounding criterion (such as when it is important that an addressee understand an utterance verbatim) or when they use media in which utterances are ephemeral and the costs of changing speakers are relatively low (such as telephone conversations, as compared to email).

Many direct manipulation interfaces support grounding rather well, with immediate visual feedback and incremental, reversible actions (Schneiderman, 1982). However, incremental grounding is not well supported in the on-line help of some applications. A user who is fortunate enough to find the answer to a question must absorb it all at once, because in order to return to the application's window, he or she must exit help (and the help window will not stay open when it is exited). This situation is particularly aggravating when the answer involves a long sequence of instructions. If the user wants to use this information step by step, she or he must repeatedly evoke and exit the help system, or else write it down or print it for later reference (which of course defeats the purpose of having an on-line help system). The solution to this problem is very simple: Enable the help window to stay open while the user returns to the application window.

MODELING HUMAN–COMPUTER INTERACTION AS JOINTLY ACHIEVED CONTRIBUTIONS

Contributions to human conversations are collective acts; that is, an utterance presented by a speaker is not part of the conversation's common ground until it has been accepted by an addressee (Clark & Schaefer, 1987, 1989; Clark & Wilkes-Gibbs, 1986). This acceptance happens through the systematic exchange of evidence during the grounding process. However, the situation is quite different in human–computer interaction. In direct manipulation interfaces, grounding often happens serendipitously, when the relevant objects are represented on the screen and are thus shared with users. In interfaces that are less graphical or entirely linguistic, shared context is much poorer unless system designers allow for appropriate feedback messages. Without systematic support for determining what kinds of messages to provide and when to provide them, the exchange of evidence is ad hoc at best. In this section I describe Clark & Schaefer's formal model of jointly achieved contributions in conversation. Then I describe proposals for incorporating these ideas into human–computer dialog (Brennan & Hulteen, 1995; Cahn & Brennan, 1997), in an attempt to solve the grounding problem in human–computer interaction.

The Contribution Model

According to the model proposed by Clark and Schaefer (1987, 1989), contributions have a *presentation phase*, in which the speaker produces an utterance addressed to a conversational partner, followed by an *acceptance phase*, in which the partner may explicitly acknowledge the utterance, modify it, clarify it, or implicitly accept it by continuing with the next relevant utterance. A particular utterance may present a contribution at the same time that it fulfills the acceptance phase for a previous contribution. Contributions may be nested within other contributions as parts of clarification subdialogs. Because an utterance presented by one partner does not become a contribution until it has been accepted by the other, both speakers and addressees are responsible for what is contributed.

Consider *A*'s utterance from an earlier example, "You do not have any nails, do you?" Clark and Schaefer (1987, 1989) described four possible states that an addressee, *B*, can be in, with respect to such an utterance, *u'*, by a speaker, *A*:

State 0: *B* did not notice that *A* uttered any *u'*.
State 1: *B* noticed that A uttered some *u'* (but was not in State 2).
State 2: *B* correctly heard *u'* (but was not in State 3).

State 3: *B* understood what *A* meant by *u'*.

In conversation, the addressee's response helps the speaker diagnose which of these states the addressee is in. According to Clark & Schaefer (1987, 1989), some utterances provide stronger evidence than others. For example, *B* may respond by repeating part or all of what he or she takes *A* to mean (strong, explicit evidence), or he or she may respond with an acknowledgment (somewhat weaker explicit evidence), or *B* may simply continue with what he or she takes to be a relevant next utterance (implicit evidence). We (Brennan & Hulteen, 1995) have extended these states in order to enable a system that uses natural language or speech to systematically provide context-appropriate feedback so that users can more easily identify the system's state with respect to a particular utterance.

Adapting The System's Feedback To Its State Of Understanding

As an example, consider a remote speech interface used over a telephone. Instead of direct manipulation, the paradigm is one of delegation to an agent. The problem is that after the user speaks, even the smallest of delays can be ambiguous. Has the system heard the utterance? Was the utterance interpretable? Or is the system still processing it? Alternatively, is it taking time for the application to carry out the command? In the face of so many possible attributions, users adopt various strategies such as repeating the utterance, hyperarticulating, shouting, or rephrasing. Some of these strategies, although they work in human conversation, may make matters worse with a speech recognizer. For instance, when people hyperarticulate, they pronounce commands in an exaggerated form that is unlike the tokens that the system was trained with, and so successful recognition is less likely than with commands pronounced naturally.

The Need for Context-Sensitive Feedback. In order to enable a user to coordinate his or her behavior with the many possible states a system can be in, the system needs to provide enough feedback for the user to recognize its state. For instance, a system can echo the user's utterance once it has been received:

User: Read email messages.
System: I heard you say *Read email messages*.

This kind of response provides the user with positive evidence that the system has attended to, heard, and interpreted the utterance. But consider the case when the evidence is negative:

User: Read email messages.
System: What was that again?

At this point, the user can tell that the system was attending and heard something, but was not able to interpret the utterance. Other negative evidence is possible when input is misrecognized:

User: Read email messages.
System: I heard you say *Delete email messages*.

Here, it is ambiguous whether the user will be able to interrupt the system in time to intercept this destructive command. When there is a high cost to misunderstanding, it is safer to respond with a request for confirmation.

User: Read email messages.
System: Do you want to *Delete email messages*?
User: No! Read email messages.
System: I heard you say *Read email messages*.

A request for confirmation lets the user intervene after a system's interpretation, before it acts on the command. Echoing an utterance and requesting confirmation both provide good evidence of whether the speech recognizer has heard the user's input. However, having to confirm each command is tiresome, especially when speech recognition performance is accurate. So it is also desirable for a system to adjust the feedback it provides depending on the dialog history; that is, if the recognition rate is high so far (the evidence being that the user has not had to initiate repairs), then the system could stop echoing all the user's commands and instead provide feedback at a higher, task level.

User: Read email messages.
System: You have five new email messages . . .

We have proposed that the feedback a system provides should be adapted to several factors: (a) the particular state the system has reached with respect to the user's utterance; (b) the likely costs of misunderstanding; (c) the dialog history so far; and (d) the ambient noise level in the user's environment (Brennan & Hulteen, 1995).

Grounding With a Spoken Dialog System. When a user delegates actions to a computer, things can go wrong in a wide variety of ways, especially when the interface acts as an intermediary between the user and one or more applications. With this in mind, we have extended Clark and Schaefer's (1989) model to cover the states that a computer system may be in with respect to a user's command (Brennan & Hulteen, 1995). From the user's perspective, these states are:

State 0: **Not attending**. The system is not able to receive input from the user.

State 1: **Attending**. The system is able to receive input (but is not yet receiving any).

State 2: **Receiving**. The system is receiving input (but the input is not yet recognized as well formed).

State 3: **Recognizing**. The system recognizes the input as well formed (but has not yet mapped it onto any plausible interpretation).

State 4: **Interpreting**. The system has reached an interpretation (but has not mapped the utterance onto an application command).

State 5: **Intending**. The system has mapped the user's input onto a command in its application domain (but has not yet acted).

State 6: **Acting**. The system attempts to carry out the command (an attempt that may or may not turn out to be successful).

State 7: **Reporting**. The system has attempted to carry out the command, and reports to the user any relevant evidence from the application domain.

In our model, States 0 to 2 follow Clark and Schaefer's, and their State 3 has been expanded into two states. States 5 to 7 are necessary extensions for dialogs that delegate actions to an agent. For some kinds of systems, the distinctions between these states may not be meaningful (that is, certain kinds of errors may not be possible); for others, it may be appropriate to divide these states into further stages. For instance, State 2 (receiving) errors are common in speech recognition interfaces where input may be heard but not parsed; they happen in text-based natural language interfaces when a word is misspelled, an unknown word is used, or keyboard input is noisy. But a speech recognition interface without a parser (one that maps a whole utterance onto a command without analyzing its constituents) would not need to distinguish between States 3 and 4.

Coordinating Two Distinct Knowledge States

Misunderstandings cannot be repaired unless they can be recognized (Cahn & Brennan, 1997; Lewis & Norman, 1986; Luperfoy & Duff, 1996). Even though two people in conversation build shared representations, there is always some asymmetry; one person invariably recognizes a problem before the other one does. In order for a problem to be recognized, each partner needs to provide feedback not only when there is a problem, but also when the partner believes there is *not* a problem. A common assumption in human–computer dialog design, however, is that a system need provide only negative evidence (i.e., an error message) when it is unable to complete processing at a particular stage. To make this assumption is to minimize the opportunities for users to recognize problems that the system cannot recognize. By the same token, there is typically no provision in the interface for a *user* to provide a *system* with negative evidence. Both of these inadequacies need to be addressed in order to support grounding in human–computer interaction.

Positive Evidence From Systems. A system should at least be able to give the user positive evidence at each meaningful point where it could break down, where meaningful is defined as wherever the user could take some action to repair or prevent a problem. Many kinds of positive evidence are possible, such as these examples:

State 1: "I'm listening."
State 2: "I heard something."
State 3: "I heard you say *Read email messages*."
State 4: "Do you want me to *Read email messages*?"
State 5: "OK, I'll *Read email messages*."
State 6: (*system reads email messages aloud*)
State 7: "That's all."

Obviously, the system should not provide feedback at *all* of these states, for that would be both tedious and redundant. Our proposal (Brennan & Hulteen, 1995) is that the system should provide negative evidence about the first state in which processing cannot be completed. For risky commands, the system should provide positive evidence at the state before which the risk occurs (here, at State 4, in the form of a request for confirmation). Positive evidence about having reached a particular state should also be provided whenever the system has recorded recent failures to reach that state or if the user has recently indicated to the system that it was in error about reaching a particular state. For instance, with a history of such problems at State 3, the system should echo the user's utterances until there is sufficient evidence (in the form of a series of successful recognitions) that these problems no longer exist. This approach assumes that evidence should be provided about the highest state at which the system has either *completed* or *attempted* processing, because reaching a particular state presumes that the system has successfully completed processing at lower-numbered states.[8] More details and examples of adaptive feedback are provided in Brennan and Hulteen (1995).

Negative Evidence From Users. In addition to enabling systems to provide positive evidence to users (Brennan & Hulteen, 1995), we have proposed that users be able to provide negative evidence to systems (Cahn & Brennan, 1997).

[8] Exceptions to this assumption are possible when two partners are capable of indirect communication—for instance, *A* may recognize *B*'s intention without hearing exactly what *B* said.

Systems typically do not provide users with any way at all to signal that they are unhappy with a particular response; users are expected to just take what they can get and go on with the next query or command. A notable exception is work by Moore (1989, 1995) which enabled users to request explanations or express a vague but perfectly valid need for more information by responding with "huh?" If the user had the ability to respond to the system with "ok" (either to explicitly accept the response or to implicitly accept it by just going on with the next query), "huh?" (to request an explanation), "no, I meant" (to initiate a repair of the last contribution), and "never mind" (to abort), this would provide a beginning for the user to negotiate acceptance of the system's responses. The system would keep a structured dialog history in the form of jointly produced contributions (Cahn & Brennan, 1997). Information presumed by the system to be in common ground would be extracted from successful contributions (those with completed acceptance phases) and represented in the dialog history, while "huh" and "no, I meant" would evoke context-sensitive subdialogs in which the content of a contribution would not be added to the dialog history until the user and system repaired the problem at hand. If they were unable to do so, the user could simply abort the contribution. Throughout this process, the dialog history would keep track of how smooth or effortful an interaction was. More detail about this proposed architecture for explicitly supporting and modeling grounding using Clark and Schaefer–style contribution trees is provided in Cahn and Brennan (1997).

CONCLUSION

In this chapter, I have discussed the grounding process and how it enables people to coordinate their distinct knowledge states, whether they are conversing face to face or electronically. I have argued that grounding is important to consider in human–*computer* dialog as well, and that many of the problems that arise when people try to use computers can be explained by inadequate feedback and impoverished context. Direct-manipulation-style interfaces that use desktop or tool-based metaphors for computing are common in today's computer systems; they tend to be easy to use because they support grounding by providing consistent and concrete representations of data, operations, and system states. Direct manipulation interfaces typically provide clear options for what a user can do next (thus bridging what Hutchins et al. [1986], termed the *gulf of execution* that exists between users and their goals), as well as feedback about a command's success or failure (bridging Hutchins et al.'s [1986] *gulf of evaluation*). By contrast, natural language and speech interfaces to applications are still relatively rare; even though speech and language technologies have improved rapidly, interfaces that depend on these technologies are typically underdetermined and often provide no explicit support for recovering from errors. This combination can be enough to make a speech or language interface unusable.

Communicating With Interface Agents

Computer interfaces are performing more and more complex tasks, such as enabling users to write programs, advising them, guiding them through simulation environments, filtering their email, teaching them, reminding them, traversing the Internet to search for specific information in their behalf, and scheduling their appointments by communicating with other users' calendars. Such applications take on more initiative than do text editors, spreadsheets, and drawing programs, and so they do not lend themselves as easily to tool metaphors; instead, they are more like coaches or assistants. As interfaces manage more complex tasks, become more "intelligent," and enable users to delegate more responsibility, the metaphor of the interface as *agent* (considered radical in the early 1980s) will become more commonplace for tasks that require delegation.

Agent-style interfaces need to have not only expertise in a particular task domain but also a general ability to communicate with users. Such communication is necessary to convey information about complex situations, to win users' confidence that the system agent is acting appropriately in their behalf, and to appropriately distribute responsibility between the users and the system. Although tool-based direct manipulation interfaces embody a theory of communication, communication in other kinds of interfaces is typically handled in an ad hoc manner. We have proposed several ways in which the grounding process can be systematically supported in agent-style interfaces, particularly when the currency of interaction is speech or language (Brennan & Hulteen, 1995; Cahn & Brennan, 1997). With an architecture that supports grounding in human–machine dialog, language-based interfaces could become as easy for the average user as direct manipulation ones.

The theoretical framework proposed by Clark and his colleagues (Brennan, 1990a; Clark, 1996; Clark & Brennan, 1991; Clark & Schaefer, 1989; Clark & Wilkes-Gibbs, 1986; Isaacs & Clark, 1987; Schober & Clark, 1989) provides a systematic way in which to model communication between two communicating agents, whether they are human or machine (Brennan & Hulteen, 1995). In everyday conversation, partners seek and provide both positive and negative evidence about beliefs, intentions, and interpretations, in order to make portions of their mental states converge (Brennan, 1990a). Many language-based interfaces provide ample *negative* evidence in the form of error messages but only minimal *positive* evidence; they also act as if most of their responses will be acceptable to users by not seeking evidence of acceptance from users and not providing any way to initiate clarification subdialogs. If language-based interfaces are to support mixed initiative dialogs (in which either user or system can flexibly take the initiative), then they need to support the systematic exchange of both positive and negative evidence.

The Synergy Between Psycholinguistics and Human–Computer Interaction

Finally, the domain of human–computer interaction is a particularly relevant application for cognitive and social psychologists who study psycholinguistics, for two reasons. First, experimental research has illuminated general principles about processing, representation, and interaction that can be applied directly to explaining, predicting, and improving human–computer interaction. Without such underlying principles, progress in interface design will be ad hoc at best, especially for multimodal, "intelligent" systems that use speech and language. At the same time, human–computer interaction provides an ideal testbed for demonstrating and testing models and principles such as the contribution model (Clark & Schaefer, 1987, 1989), the principle of least collaborative effort (Clark & Wilkes-Gibbs, 1986), and the costs and tradeoffs of grounding in different media (Clark & Brennan, 1991). Transporting models from social and cognitive psychology to electronic communication and embodying such models in software has the potential to bring additional clarity and pragmatism to these fields.

ACKNOWLEDGMENTS

I thank my collaborators: Heather Bortfeld, Janet Cahn, Herbert Clark, Eric Hulteen, Gregory Lee, Justina Ohaeri, Pamela Stellmann Reis, Claire Rubman, and especially, Michael Schober, who also provided valuable comments on this chapter. I am also grateful to the very patient editors of this volume, Susan Fussell and Roger Kreuz. This material is based upon work supported by the National Science Foundation under Grants IRI9202458 and IRI9402167 and by Apple Computer, Inc. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation or of Apple Computer, Inc.

REFERENCES

Bortfeld, H., & Brennan, S. E. (1997). Use and acquisition of idiomatic expressions in referring by native and non-native speakers. *Discourse Processes, 23,* 119-147.

Brennan, S. E. (1988). The multimedia articulation of answers in a natural language database query system. In *Proceedings, Second Conference on Applied Natural Language Processing* (pp. 1–8). Austin, TX: Association for Computational Linguistics.

Brennan, S. E. (1990a). *Seeking and providing evidence for mutual understanding*. Unpublished doctoral dissertation, Stanford University, Stanford, CA.

Brennan, S. E. (1990b). Conversation as direct manipulation: An iconoclastic view. In B. K. Laurel (Ed.), *The art of human–computer interface design* (pp. 393–404). Reading, MA: Addison-Wesley.

Brennan, S. E. (1991). Conversation with and through computers. *User Modeling and User-Adapted Interaction*, *1*, 67–86.

Brennan, S. E. (1996, October). Lexical entrainment in spontaneous dialog. In *Proceedings, ISSD 96, International Symposium on Spoken Dialog* (pp. 41–44). Philadelphia, PA: Acoustical Society of Japan.

Brennan, S. E. (in press). The vocabulary problem in spoken dialog systems. In S. Luperfoy (Ed.), *Automated Spoken Dialog Systems.* Cambridge, MA: MIT Press.

Brennan, S. E., & Clark, H. H. (1996). Lexical choice and conceptual pacts in conversation. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *22*, 1482–1493.

Brennan, S. E., & Hulteen, E. (1995). Interaction and feedback in a spoken language system: A theoretical framework. *Knowledge-Based Systems, 8,* 143–151.

Brennan, S. E., & Ohaeri, J. O. (1994). Effects of message style on user's attributions toward agents. In *Conference Companion, CHI '94, Human Factors in Computing Systems* (pp. 281–282). Boston, MA: ACM Press.

Brennan, S. E., & Williams, M. (1995). The feeling of another's knowing: Prosody and filled pauses as cues to listeners about the metacognitive states of speakers. *Journal of Memory and Language, 34,* 383–398.

Brown, P., & Levinson, S. C. (1987). *Politeness: Some universals in language usage*. Cambridge, England: Cambridge University Press.

Cahn, J. E., & Brennan, S. E. (1997). *A computational architecture for the progression of mutual understanding in dialog.* Manuscript submitted for publication.

Clark, H. H. (1996). *Using language*. Cambridge, England: Cambridge University Press.

Clark, H. H., & Brennan, S. E. (1991). Grounding in communication. In L. B. Resnick, J. Levine, & S. D. Behrend (Eds.), *Perspectives on socially shared cognition* (pp. 127–149). Washington, DC: American Psychological Association.

Clark, H. H., & Marshall, C. R. (1981). Definite reference and mutual knowledge. In A. K. Joshi, B. Webber, & I. A. Sag (Eds.), *Elements of discourse understanding* (pp. 10–63). Cambridge, England: Cambridge University Press.

Clark, H. H., & Schaefer, E. F. (1987). Collaborating on contributions to conversations. *Language and Cognitive Processes*, *2*, 19–41.

Clark, H. H., & Schaefer, E. F. (1989). Contributing to discourse. *Cognitive Science, 13*, 259–294.

Clark, H. H., & Wilkes-Gibbs, D. (1986). Referring as a collaborative process. *Cognition, 22*, 1–39.

Cohen, P. R. (1984). The pragmatics of referring and the modality of communication. *Computational Linguistics, 10,* 97–146.

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* (pp. 339–352). Hillsdale, NJ: Lawrence Erlbaum Associates.

Gibson, J. J. (1977). The theory of affordances. In R. E. Shaw & J. Bransford (Eds.), *Perceiving, acting, and knowing*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Holtgraves, T. M. (1998). Interpersonal foundations of conversational indirectness. In S. R. Fussell & R. J. Kreuz, (Eds.), *Social and cognitive approaches to interpersonal communication.* Hillsdale, NJ: Lawrence Erlbaum Associates

Hutchins, E. L., Hollan, J. D., & Norman, D. A. (1986). Direct manipulation interfaces. In D. A. Norman & S. W. Draper (Eds.), *User centered system design* (pp. 87–124). Hillsdale, NJ: Lawrence Erlbaum Associates.

Isaacs, E. & Clark, H. H. (1987). Reference in conversation between experts and novices. *Journal of Experimental Psychology: General, 116,* 26–37.

Jefferson, G. (1972). Side sequences. In D. Sudnow (Ed.), *Studies in social interaction* (pp. 294–338). New York: Free Press.

Krauss, R. M. & Weinheimer, S. (1966). Concurrent feedback, confirmation, and the encoding of referents in verbal communication. *Journal of Personality and Social Psychology, 4,* 343–346.

Laurel, B. (1990). Interface agents: Metaphors with character. In B. Laurel (Ed.), *The art of human–computer interface design* (pp. 355–366). Reading, MA: Addison-Wesley.

Lewis, C., & Norman, D. A. (1986). Designing for error. In D. A. Norman & S. W. Draper (Eds.), *User centered system design* (pp. 411–432). Hillsdale, NJ: Lawrence Erlbaum Associates.

Luperfoy, S. & Duff, D. (1996). A centralized troubleshooting mechanism for a spoken dialogue interface to a simulation application. *Proceedings, ISSD 96, International Symposium on Spoken Dialog* (pp. 77–80). Philadelphia, PA: Acoustical Society of Japan.

Moore, J. D. (1989). Responding to "Huh?": Answering vaguely articulated follow-up questions. *Proceedings, CHI '89, Human Factors in Computing Systems*. (pp. 91–96). Austin, TX: ACM Press.

Moore, J. D. (1995). *Participating in explanatory dialogues*. Cambridge, MA: MIT Press.

Norman, Donald A. (1990). *The Design of Everyday Things*. New York: Doubleday.

Ochsman, R. B. & Chapanis, A. (1974). The effects of 10 communication modes on the behavior of teams during cooperative problem-solving. *International Journal of Man–Machine Studies, 6*, 579–619.

Reichman, R. (1986). Communication paradigms for a window system. In D. A. Norman & S. W. Draper (Eds.), *User centered system design* (pp. 285–313). Hillsdale, NJ: Lawrence Erlbaum Associates.

Rheingold, H. (1985). *Tools for thought: The people and ideas behind the next computer revolution.* New York: Simon & Schuster.

Schmandt, C., Arons, B., & Simmons, C. (1985). Voice interaction in an integrated office and telecommunications environment. In *Proceedings, 1985 AVIOS, American Voice I/O Society* (pp. 51–57). San Jose, CA: American Voice I/O Society.

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,* 57–69.

Shneiderman, B. (1986). *Designing the user interface: Strategies for effective human–computer interaction* (1st ed.). Reading, MA: Addison-Wesley.

Shneiderman, B. (1992). *Designing the user interface: Strategies for effective human–computer interaction* (2nd ed.). Reading, MA: Addison-Wesley.

Shneiderman, B. (1993, January). Beyond intelligent machines: Just do it! *IEEE Software,* 100–103.

Schober, M. F., & Clark, H. H. (1989). Understanding by addressees and overhearers. *Cognitive Psychology, 21,* 211–232.

Smith, V. L. & Clark, H. H. (1993). On the course of answering questions. *Journal of Memory and Language, 32,* 25–38.

Sutherland, I. E. (1963). *Sketchpad: A man–machine graphical communication system.* MIT. Lincoln Laboratory Technical Report (No. 296) Lexington, MA: MIT Lincoln Laboratory.

Whittaker, S. J., Brennan, S. E., & Clark, H. H. (1991). Coordinating activity: An analysis of interaction in computer-supported cooperative work. In *Proceedings, CHI '91, Human Factors in Computing Systems* (pp. 361–367). New Orleans, LA: Addison-Wesley.

Wilkes-Gibbs, D. (1986). *Collaborative processes of language use in conversation*. Unpublished doctoral dissertation, Stanford University, Stanford, CA.

Wilkes-Gibbs, D., & Clark, H. H. (1992). Coordinating beliefs in conversation. *Journal of Memory and Language, 31,* 183–194.

Yngve, V. H. (1970). On getting a word in edgewise. In *Papers from the sixth regional meeting of the Chicago Linguistic Society* (pp. 567–578). Chicago: Chicago Linguistic Institute.