Where the Action Is

Where the Action Is

The Foundations of Embodied Interaction

Paul Dourish

A Bradford Book
The MIT Press
Cambridge, Massachusetts
London, England

1 A History of Interaction

It is a truism that computers are becoming faster and more powerful all the time. They play an ever larger role in our lives, giving us access to more and more information, being incorporated into more and more of our devices, and creating whole new forms of interaction and activity that we would never otherwise have imagined. From desktop computers to laptops to personal digital assistants, not to mention bank teller machines, microwave ovens, cellular telephones, and ticket machines, we encounter computers in all aspects of everyday life. The ever-expanding province of computation is a commonplace, the topic of a million coffee-shop conversations, television reports, and newspaper headlines. We talk about how fast it is changing, but we talk much less about the ways in which it is not. Many things about computers are not changing at all. Our basic ideas about what a computer is, what it does, and how it does it, for instance, have hardly changed for decades. Nor have the difficulties we encounter actually using computers.

Our experience using computers reflects a trade-off that was made fifty years ago or more. When computers were first being developed commercially, they were extremely expensive devices. Computer time was much more expensive than your time or mine. In that context, efficiency dictated that we minimize the amount of computer time any job or activity needed, even if that meant burdening the people who wanted to submit the job. If a rigid, formalized input language was easier for the system to process, for example, then the cost in people's time to format their data in that language was more than offset by the savings in processing time that would result. Because most uses of computers were military and commercial rather than personal, it was hard to disagree with this sort of economic argument. It gave rise to a model that favors performance over convenience, and places a premium on the computer's time rather than people's time. This model is still with us today.

However, in light of those commonly observed transformations in computer power, we are now in a position to reconsider the trade-off. Arguably, we *must*. Computers are now so much faster and more powerful, giving us access to so much more information that we are simply no longer able to manage and assimilate it. At the same time, those powerful computers spend 95 percent of their time doing absolutely nothing. Modern personal computers perform very few tasks that use their full capacity for longer than a second or two. Outside these brief bursts of activity, most of the time they do nothing at all, generally while we try to figure out what to make of what just happened or what we want to do next.

At the same time, we increasingly see computers incorporated into devices other than the traditional PC sitting on the desk. Computation is part of your cellular telephone, your microwave oven, your car, and a host of other technologies. The rise of so-called embedded computing reflects the fact that computation can be usefully harnessed for more than just traditional desktop computing. It can also help us as we get up and move about in the world, which we generally do more of than sitting at desks (or would, if the computers didn't shackle us to them). However, this new form of computation exacerbates the effects of the trade-off between the work that the user and the system do. As I sit at my desktop computer, it occupies the whole of my attention; but that would be a terrible idea in a computer I'm using while driving, or crossing the street, or trying to enjoy a conversation with friends.

These two trends—the massive increase in computational power and the expanding context in which we put that power to use—both suggest that we need new ways of interacting with computers, ways that are better tuned to our needs and abilities. Over the last few years, research into Human-Computer Interaction (HCI) has begun to explore ways to control and interact with a new breed of computer systems. Prototype systems have been developed; new forms of interaction explored; new research groups established; new designs developed and tested.

This book is a contribution to the emerging literature on this new approach to interacting with computers, one that I call "Embodied Interaction." Embodied Interaction is interaction with computer systems that occupy our world, a world of physical and social reality, and that exploit this fact in how they interact with us.

There are two ways in which the material I want to present in this book differs from other explorations in HCI. The first difference concerns the set of entities that will appear here. In particular, although computer interfaces are the general topic, interfaces themselves will not appear too often. Here, I am more concerned with interaction than I am with interfaces, and more concerned with computation than I am with computers. When I say that I am more concerned with interaction than with interfaces, I mean that I will be dealing with the ways in which interactive systems are manifest in our environment and are incorporated into our everyday activities, rather than with the specific design of one user interface or another. Similarly, when I say that I am more concerned with computation than with computers, I mean that I want to address the idea of computation per se—of active representations embodied in hardware and software systems—rather than the specific capabilities of systems available at the start of the new millennium. So, gigabytes and megahertz will not be at issue, but representational power will be.

The second difference is in the way that those topics will be addressed. In particular, as you might guess on the basis of my concern with interaction and computation, I want to address a set of topics that are more foundational than technical. This is not a source book of design solutions, or a how-to manual for interface developers—although these practical matters will certainly arise, and I hope that designers will find something useful here. In fact, the very reason for exploring foundations is to support the design and evaluation of new systems, tools, and interaction modalities. The goal of this foundational exploration is to provide resources to designers and system developers, by giving them tools they can use to understand and analyze their designs.

Traditionally, the central component of any account of computation has been algorithms or procedures—step-by-step models that specify the sequential behavior of a computer system. In turn, because they are based on an analogy between mental phenomena and computation, cognitive science and AI have also predominantly espoused a step-by-step model of procedural execution. In the last few years, though, this procedural approach has been challenged by a new conceptualization of computational phenomena that places the emphasis not on *procedures* but on *interaction* (Wegner 1997). Interactional approaches conceptualize computation as the interplay between different components, rather than the fixed and prespecified paths that a single, monolithic computational engine might follow. These models of computation have more in common with ecosystems than with the vast mechanisms we used to imagine. They emphasize diversity and specialization rather than unity and generality. Perhaps there is, in this, something of the spirit of the times; perhaps, too, the rise of new computational paradigms such as parallel systems, object-oriented programming, and Internet-style software design is implicated in this change. The change, though, has occurred across a wide range of areas of computational investigation. It has affected how we think about computation from a mathematical perspective, leading to new theoretical accounts of systems such as Hoare's CSP (Hoare 1985) or Milner's work on CCS and the Pi Calculus (Milner 1995, 2000); it has affected how we think about computational models of mind, as reflected by Minsky's "Society of Mind" (Minsky 1988), Agre's critique of computational reasoning (Agre 1997), or Brooks's approach to robotics (Brooks 1999); and it has led to new accounts of the practice of programming (Stein 1998).

You might think that studies of how people use computers must always have been built around a model of the world that gives pride of place to interaction, but in fact HCI has traditionally been built on a procedural foundation. HCI, from its very beginning, took on the trappings of the traditional computational model and set out its account of the world in terms of plans, procedures, tasks, and goals. In contrast, the model of HCI I set out here is one that places interaction at the center of the picture. By this I mean that it considers interaction not only as *what* is being done, but also as *how* it is being done. Interaction is the means by which work is accomplished, dynamically and in context.

Some background will help to clarify what this means and to set the stage for the argument this book will develop. The context is the historical evolution of the idea of interaction and the technology of HCI.

A Historical Model of Interaction

Just as computers have evolved considerably in their short history, so have styles of human-computer interaction. There are many ways to conceptualize the history of interaction with computer systems. The purely technological view, for example, would recount the history of the input and output devices that have characterized different stages of interface development, and would describe their computational demands. A political view would consider the movement of ideas from one laboratory to another as researchers respond to the demands and interests of funding agencies and so forth, while an economic view would consider how user interface development has influenced, and been influenced by, the growth of the high-tech industry and PC economy. Grudin (1990) describes the history of interaction as the story of the "computer reaching out," in which interaction moves from being directly focused on the physical machine to incorporating more and more of the user's world and the social setting in which the user is embedded. Although Grudin's analysis is now a decade old, it is interesting to see the ways in which later trends in HCI design—including some that are of particular interest in this book—have followed quite closely the directions that he laid out.

I want to explore a slightly different view here, in order to set some context for the discussion that will follow. In particular, I want to present the stages in the historical development of user interfaces in terms of the different sets of human skills they are designed to exploit. This is not a different history of HCI, of course, but merely a different telling of the history, with the emphasis in a slightly different place. As is perhaps appropriate for a discipline that concerns itself as much with human abilities as with technological opportunities, it draws attention to the human experience of computation. The are four separate phases of development to discuss. I characterize them as electrical, symbolic, textual, and graphical forms of interaction.

Electrical

Today, when we talk of "computers," we invariably mean digital devices. The computer as we know it is inescapably bound up with the ones and zeros of digital logic. It was not always this way. Originally, the word "computers" referred to human beings—people whose daily work was the figuring of calculations, such as for producing engineering tables. However, even when "computers" became electronic devices, they were not necessarily digital ones. Before digital computers came analog computers. Analog computers did not rely on the discrete logic that characterizes modern computing devices; instead, they relied on the use of standard components such as resistors and capacitors to create electronic models of continuous natural phenomena (such as wave motion, the interaction of electronic forces, or the movements of objects under gravity). Essentially, the analog computer was the apparatus for laboratory simulations that took place not in the physical world, but in an analogous electronic reality. To set up a new experiment, the machine would have to be reconfigured, possibly quite radically, through the incorporation of new circuits. This task-specificity was shared by the early digital computers, too. Even after we had made the move from analog electronics to digital logic, the earliest digital computers were special purpose devices, designed as automatic calculators to solve specific problems—often, inevitably, in military domains (such as calculating missile trajectories or exploring patterns in coded messages).

Although there is some debate about precisely who was the first to make the move—perhaps Eckert and Mauchley with EDVAC in Philadelphia, or Williams and Kilburn building the Small-Scale Experimental Machine, known as "Baby," at Manchester, or one of the other contenders—what *is* generally accepted is that the critical development in digital computing was that of the *stored program computer.* In contrast to earlier designs, a stored program computer is a machine whose operation is not directly encoded in its circuits, but rather is determined by a sequence of instructions held in its memory—instructions that can, clearly, be changed or replaced much more easily than the electrical circuits could be reconfigured. Nonetheless, the first age of computing, around the time that this transition took place, relied heavily on an understanding of the electronics that made up any given machine. Every machine was a prototype; every program, uniquely designed for a specific computer (and perhaps even a specific version or configuration of that computer). What we currently refer to as "instruction sets"—the set of low-level operations that processors such as the Pentium or PowerPC can understand—were, at

that stage in the history of computation, intimately tied to the individual details of the circuitry of any particular computer. So, even as we made the transition from hardware configuration to digitally stored programs, the dominant paradigm for interaction with the computer was electronic. Entering a new program, even if that program was to be stored digitally in the memory of the computer, could still bear a remarkable resemblance to electronic reconfiguration, involving plugboards and patch cables. Indeed, such programming activity was often accompanied with the development of new circuits that could extend the operation of the system. The boundary that we now take for granted between hardware and software was much fuzzier then; interacting with the system, and developing new programs, relied on a thorough understanding of the electronic design.

Symbolic

The next stage of development is characterized by the emergence of symbolic forms of interaction. The movement from one stage to another is not a sudden and clear transition; instead, it is a general trend that emerges in a number of different ways. We can see it in the basic models offered for programming systems, which was the primary form of interaction between human and computer at a time when "users" as we now know them did not yet exist.

As the transition from electrical to symbolic approaches gradually took hold, programming computers came to require less understanding of the detailed construction of each particular machine, and relied increasingly on regularized and well-understood capacities that would be available across a wide range of machines—register files, index registers, accumulators, and so forth. At the same time, the primary form of programs moved from a numeric form (that is, the "machine language" of raw instructions that a machine would understand) to other symbolic forms that were more readily understandable to human beings. So-called assembly languages are essentially symbolic forms of machine language, using mnemonic codes that stand in one-to-one correspondence with the machine level instructions, so that a sequence of instruction codes such as "a9 62 82 2c" is rendered as a symbolic expression such as "movl (r1+), r2."[1]

Since assembly languages are simply a different rendering of machine languages—symbolic forms that describe sets of specific instructions—they are just as tied as machine languages to particular systems, although, by this stage, computer systems were being produced industrially rather than developed as one-off prototypes in laboratories. But they are in no way portable between machines of different sorts, even today—assembly programs for an Intel processor yield machine instructions that will run only on Intel processors, and not on other processors made by Motorola. A further progression along the symbolic path, though, came with the development of the early programming languages such as LISP and FORTRAN. Essentially, these lay down two sets of rules. The first set describes what structural properties a set of instructions will have to be valid programs—what rules must be followed when creating something that is a FORTRAN program rather than simply gibberish. The second describe how programs can be turned into a set of (machine language) instructions for the computer to execute. The important point is that, whereas programs would previously be specified with relation to a specific machine language (perhaps encoded as assembly instructions, but still tied to a particular sort of computer), the programmer's activity was now lifted to a more abstract level that was simultaneously a more natural form of expression and independent of the precise details of any specific computer, its implementation and configuration.

The introduction of programming systems such as assemblers and programming languages moved computer interaction, then, from an electronic level to a symbolic one. It introduced a set of symbolic representations of computer system operation as the primary modality by which interaction was conducted. Interestingly, this was also reflected in the physical interaction with systems. Punched cards, for example, can be regarded as a primitive form of symbolic interaction, especially because punched card systems quickly came to incorporate both *data* cards (that is, cards that carried information for programs to process) and *control* cards (instructing the system to begin and end jobs, etc.) The control cards, then, provide a symbolic language for controlling the behavior of the system.

The reason I want to cast the history of interactive computing in terms of these different sorts of interaction modalities is that it draws our attention to the fact that they exploit quite different sets of skills. We are all highly skilled at various forms of symbolic interaction; language and communication, for us, are largely symbolic in nature, whether these symbols take the forms of icons, traffic signs, flags, maps, or marks on paper. Symbolic interaction is a much more natural and intuitive form of interaction for us than the electronic form that had previously been necessary; and it allows us to bring to bear a much more powerful set of intuitions and abilities to the interactive task. So, finding errors in assembly language programs is much less error-prone than trying to do the same in machine language; and debugging programs written in so-called high level languages is easier still (although, as any programmer will tell you, it is still the most time-consuming and intricate part of the process of developing software). We are generally able to exploit a greater range of skills—visual, cognitive, and so on—as we move from electrical to symbolic forms of interaction.

Textual

The best-developed form of symbolic interaction with which we are familiar is, of course, written language and textual interaction. So it is only natural that symbolic interaction with computers should gradually extend into the textual domain.

Of course, most of the examples I provided for symbolic interaction were textual in nature, one way or another. For my purposes, a distinction can be made between symbolic and textual interaction by looking at the *actual interaction* with the computer. So, although programs written in assembly language are clearly textual, the form in which they arrive at the computer might not be textual at all, but might be encoded on punched cards or other symbolic media. However, the modes of interaction with technology are continually shifting as technology develops and new opportunities present themselves, and before long the primary form of direct interaction with computers was, indeed, textual interaction, at teletype machines and video terminals.

When this transition took place, textual interaction was no longer simply a means to describe computer operations, but became the primary form of interaction. Arguably, this is the origin of "interactive" computing, because textual interfaces also meant the appearance of the "interactive loop," in which interaction became an endless back-and-forth of instruction and response between user and system. Even in these days of graphical and virtual reality interfaces, this model is still often the only recourse for some operations.

One reason that textual interaction remains so powerful is that it draws not only on the use of textual characters but on how those characters can be combined into words and sets of words. In other words, along with textual interaction came a "grammar" of interaction, one that broke input text into commands, parameters, arguments, and options. So, just as the move from electrical to symbolic interaction meant that interface designers could draw upon a new set of human skills and abilities, so too did textual interaction. Textual interaction can draw on our linguistic skills, not by letting us simply "talk" to computers (at least, outside of science fiction films), but rather by drawing on our abilities to create meaningful sentences by combining elements each of which contributes to the sense of the whole.

The compositional character of textual interaction has proven hard to replace as interfaces have developed. The value, as we will see, of later interaction modalities such as graphical user interfaces is that they make the abstract entities of computation into "real," individuable objects supporting direct interaction. However, because our programs are still constructed in terms of abstract entities, textual interaction still proves its value by giving us the ability to create instructions that operate in terms of generalities—loops, conditions, patterns, and more.

The other significant feature of the textual interface paradigm is that it brought the idea of "interaction" to the fore. Textual interaction drew upon language much more explicitly than before, and at the same time it was accompanied by a transition to a new model of computing, in which a user would actually sit in front of a computer terminal, entering commands and reading responses. With this combination of language use and direct interaction, it was natural to look on the result as a "conversation" or "dialogue." These days, this idea of dialogue is central to our notion of "interaction" with the computer, replacing configuration, programming, or the other ideas that had largely characterized the interplay

between users and systems in the past. So, although the notion of "interaction" with computers had important predecessors before this period—such as Ivan Sutherland's hugely influential work on Sketchpad (Sutherland 1963)—it was arguably from the paradigm of text-based dialogue that people drew the idea of "interacting with the machine." And interacting was something that we already knew how to do.

Graphical

Probably the most significant transition, in terms of the development of the user interface models that are familiar to us today, was the transition from textual to graphical interaction. Graphical interaction developed from the work of many people, including Sketchpad on the TX-2 (Sutherland 1963), and the work of Alan Kay and his colleagues at PARC, based in turn on the developmental psychology of Piaget, Bruner, and others (Kay 1993).

Just as the move from symbolic to textual interaction did more than simply replace one symbolic language with another, the move from textual to graphical interaction did not simply replace words with icons, but instead opened up whole new dimensions for interaction—quite literally, in fact, by turning interaction into something that happened in a two-dimensional space rather than a one-dimensional stream of characters. Traditional textual interaction took place at teletype machines or serial terminals, where information appeared at the bottom of the screen and scrolled up to disappear off the top. The user's input and the system's output together formed a single stream of information, arranged linearly, character by character. In contrast, graphical interaction is characterized by its use of space; information is spread out over a larger screen area, so that the locus of action and attention can move around the screen from place to place or can even be in multiple places simultaneously (e.g., in different windows). The task of managing information becomes one of managing space.

Moving from one-dimensional to two-dimensional interaction made it possible, again, to exploit further areas of human ability as part of the interactive experience. These included:

Peripheral Attention Distributing information around a two-dimensional space allows us to arrange it so that it can be selectively attended to.

For example, many applications divide the screen (or window) into two areas—a large area taking up most of the space in which the primary interaction takes place, and a smaller area, at one edge or off to the side, in which messages are displayed about the current progress of other tasks, or other ancillary information. My word processor uses this approach. It has a status bar at the bottom of the screen that shows when the document is being updated, saved, printed, and so forth and provides various pieces of information that might be helpful in managing my activity but are not central to it. By placing them in the periphery, the application exploits my ability to focus on one area while passively attending to other activity in the edge of my visual field.

Pattern Recognition and Spatial Reasoning Laying out information in two dimensions lets us apply the skills we use managing visual information in the everyday environment. Actions as simple as walking across the room or picking up a cup involve spatial reasoning skills, and these can be exploited in two-dimensional interfaces. In particular, our ability to recognize patterns in the spatial organization of information provides new ways to convey information, and opportunities to arrange data elements so that they convey information as a whole. The same techniques that allow graphs, charts, and other visual information designs to provide insight into collections of information can also be exploited when we move computational information and interaction into a two-dimensional space.

Information Density Pattern recognition draws upon the way in which certain arrangements of data can draw attention to patterns and other items of "meta-information." In turn, this raises a question of "information density." Some information can be conveyed more succinctly in graphical form than in lists of numbers or other textual representations. A picture really can be worth a thousand words; it can often be displayed more compactly and apprehended more rapidly than can its thousand-word equivalent. Of course, there are also forms of information for which a textual presentation is either desirable or required, but graphical interaction has never been *purely* graphical; instead, it extends the vocabulary of interaction to incorporate graphical as well as textual presentation forms and allows textual information to be presented within a framework that incorporates graphical elements and two-dimensional layout.

Visual Metaphors As well as giving new ways to depict data, the graphical approach can also add value by providing new ways to represent actions and the context in which actions take place. This leads to the development of visual metaphors for information management. The most widespread is the office or desktop metaphor, in which information management tasks are based around a metaphorical model incorporating filing cabinets and trashcans, graphically displayed on the screen along with the basic data elements, and so conveying a sense of the activities that can be performed over the data. In more recent systems, this has been extended. General Magic's "Magic Cap" interface, used a metaphorical depiction of an office featuring a desk (along with various desktop tools), a telephone, and a door open to a world outside; note-taking applications often feature graphical depictions of notebooks or index cards; and so on.

The development of graphical interaction techniques led to a model of interface design known as *direct manipulation,* in which these elements are combined and extended. The fundamental principle in direct manipulation interfaces is to represent explicitly the objects that users will deal with and to allow users to operate on these objects directly. Uploading a file to a server by naming it, or even by selecting it from an "open file" dialog, is not a direct manipulation approach; direct manipulation would advocate selecting the file icon, dragging it and dropping it onto a representation of the server. The direct manipulation style of interface extends the idea of the visual metaphor to a richer model in which the abstract objects that make up the system's conceptual model—be they records, files, connections, servers, transactions, or whatever—are realized in a metaphorical world that also defines how they interact with each other. From these separate elements, the designer builds an inhabited world in which users act. Direct manipulation interfaces exploit and extend the benefits of graphical interaction. Because the system can be controlled entirely through the manipulation of on-screen objects, all opportunities for action are "out in the open." This eliminates (or, at least, reduces) the need for long sequences of action, paths that might be difficult to recognize or hard to follow.

Progress

It has been a long transition from interacting with computers using a soldering iron to interacting using a mouse. It has been neither smooth nor planned. Instead, the evolution of interaction models has gone hand in hand with the evolution of technologies, models of computation, and perceptions of the roles that computers will play in our lives.

Despite the rather chaotic evolution of interaction, it is still possible to draw out some general trends. The trend I have emphasized here is the gradual incorporation of a wider range of human skills and abilities. This allows computation to be made ever more widely accessible to people without requiring extensive training, and to be more easily integrated into our daily lives by reducing the complexity of those interactions. The "skills and abilities" perspective also offers a model for what sorts of opportunities new research directions might offer.

New Models for Interactive System Design

Graphical interaction remains the dominant paradigm for interaction with computers. In 1981 Xerox's Star was the first personal computer to ship with the features of a graphical user interface as we recognize them today—windows, menus, and a mouse—and the Macintosh, three years later, was the first to ship in volume at an affordable price. Perhaps more significantly, the release of Macintosh signaled a sea change in the way in which we interacted with computers. It simply became clear that this new paradigm was how we would interact with computers from then on.[2] Other manufacturers started shipping their machines with mice and with displays capable of supporting windowed interfaces, and the graphical user interface became the familiar face of computing.

Twenty years later, this is still true. As I write this, there are four computers here in my office, running three different operating systems; but they all display similar graphical user interfaces comprising windows, menus, and widgets such as buttons and scroll bars, controlled by a mouse sitting next to the keyboard. Although the Macintosh is arguably

the only one that was designed that way from Day 1, the style that it introduced has remained largely unchallenged. In fact, the graphical interface predominates even in those areas where its application is more questionable, from wall-sized electronic whiteboards to small handheld computers.

However, recent research programs have begun to explore new paradigms for interaction and interactive system design. Some of these will be the topics of the next few chapters, but a quick sketch is in order here.

Tangible and Social Approaches to Computing

This chapter opened by discussing how we are increasingly encountering computation that moves beyond the traditional confines of the desk and attempts to incorporate itself more richly into our daily experience of the physical and social world. Each of these areas—physical and social—has been a focus of research attention.

Work on physical interaction has been a particularly active topic in the last few years. A variety of terms have been used to encompass the different activities being carried out and concerns being addressed. I use "tangible computing" here as an umbrella term.[3]

Tangible computing encompasses a number of different activities. One general trend is to distribute computation across a variety of devices, which are spread throughout the physical environment and are sensitive to their location and their proximity to other devices. In these sorts of environments, printers and fax machines might advertise their presence to handheld computers, which can then reconfigure themselves around the set of services available in the local environment; or tags identifying individuals might signal their presence to each other so that their wearers can find out which people in a meeting room share their interests, or even just who the people are. A second trend is to augment the everyday world with computational power, so that pieces of paper, cups, pens, ornaments, and toys can be made active entities that respond to their environment and people's activities. A toy might know when it has been picked up and change the computer display to reflect the fact that its owner is clearly feeling more playful rather than concentrating on work. Or picking up a piece of paper might cause my computer to show me related documents or remind me about other things I was working on when I last worked on it. A third topic of investigation in tangible computing is how these sorts of approaches can be harnessed to create environments for computational activity in which we interact directly through physical artifacts rather than traditional graphical interfaces and interface devices such as mice. Mice provide only simple information about movement in two dimensions, while in the everyday world we can manipulate many objects at once, using both hands and three dimensions to arrange the environment for our purposes and the activities at hand. A child playing with blocks engages with them in quite different ways than we could provide in a screen-based virtual equivalent; so tangible computing is exploring how to get the computer "out of the way" and provide people with a much more direct—tangible—interaction experience.

Although perhaps less focused as a research activity than tangible computing, the last decade or so has also seen increasing attempts to incorporate understandings of the social world into interactive systems. By analogy with tangible computing, I refer to this as "social computing."

Again, it encompasses a range of different activities that are more or less aligned. One set of activities involves incorporating social understandings into the design of interaction itself. That is, it attempts to understand how the "dialogue" between users and computers can be seen as similar and dissimilar to the way in which we interact with each other. Social science offers models of social action and the establishment of social meaning, which provide insight into the design of interaction with software systems. At the same time, anthropological and sociological approaches have been applied to uncovering the mechanisms through which people organize their activity, and the role that social and organizational settings play in this process. These investigations have yielded both prototype systems and generalized understandings of the influence that social and organizational settings can have on the organization of activities around computer systems. Finally, here, a third set of investigations has explored how what we normally consider to be "single-user" interaction—one person sitting in front of one computer—can be enhanced by incorporating information about others and the activity of others. This information can, in turn, assist individuals in exploring the electronic world of a computer application in the same way that the real

world reveals to us signs and indications of the activities of others that can help us find our way around and carry on our actions—whether by "following the crowd" to find an event, sizing up the clientele when deciding on a restaurant, or knowing that a hotel is a good place to catch a taxi.

These are brief sketches of research areas, to be explored in more detail later on. However, even these overviews show that Human-Computer Interaction research is responding to the challenges of computation that inhabits our world, rather than forcing us to inhabit its own.

From Tangible and Social Computing to Embodied Interaction

My reason for viewing the history of interaction as a gradual expansion of the range of human skills and abilities that can be incorporated into interaction with computers is that I believe that it provides a valuable perspective on activities such as tangible and social computing. In particular, it shows that these two areas draw on *the same* sets of skills and abilities. Tangible and social computing are arguably aspects of one and the same research program.

This is the hypothesis that this book sets out to explore. The rest of the book will discuss the hypothesis and its implications in more detail, but I will set the argument out briefly here. It has four parts.

First, I want to argue that social and tangible interaction are based on the same underlying principles. This is not to deny their obvious differences, both in the approaches they adopt and the ways in which they apply to the design of interactive systems. Nonetheless, they share some important elements in common. In particular, they both exploit our familiarity and facility with the everyday world—whether it is a world of social interaction or physical artifacts. This role of the everyday world here is more than simply the metaphorical approach used in traditional graphical interface design. It's not simply a new way of using ideas like desktops, windows, and buttons to make computation accessible. Instead of drawing on artifacts in the everyday world, it draws on *the way the everyday world works* or, perhaps more accurately, *the ways we experience the everyday world*. Both approaches draw on the fact that the ways in which we experience the world are through directly interacting with it, and that we act in the world by exploring the opportunities for action that it provides to us—whether through its physical configuration, or through socially constructed meanings. In other words, they share an understanding that you cannot separate the individual from the world in which that individual lives and acts.

This comes about in contrast to a narrowly cognitive perspective that, for some time, dominated the thinking of computer system designers and still persists to a considerable degree. The positivist, Cartesian "naive cognitivism" approach makes a strong separation between, on the one hand, the mind as the seat of consciousness and rational decision making, with an abstract model of the world that can be operated upon to form plans of action; and, on the other, the objective, external world as a largely stable collection of objects and events to be observed and manipulated according to the internal mental states of the individual. From this perspective, a disembodied brain could think about the world just as we do, although it might lack the ability to affect it by acting in it. In contrast, the new perspective on which tangible and social computing rest argues that a disembodied brain could not experience the world in the same ways that we do, because our experience of the world is intimately tied to the ways in which we act in it. Physically, our experiences cannot be separated from the reality of our bodily presence in the world; and socially, too, the same relationship holds because our nature as social beings is based on the ways in which we act and interact, in real time, all the time. So, just as this perspective argues that we act in the world by exploring its physical affordances, it also argues that our social actions are ones that we jointly construct as we go along. A conversation between two people is shaped in response to the moment rather than abstractly planned, in much the same way as a juggler has to respond dynamically to the way in which each ball falls.

This leads to the second part of my argument, which is that the central element of this alternative perspective is the idea of *embodiment*. By embodiment, I do not mean simply physical reality, although that is often one way in which it appears. Embodiment, instead, denotes a form of participative status. Embodiment is about the fact that things are embedded in the world, and the ways in which their reality depends on being embedded. So it applies to spoken conversations just as much as to

apples or bookshelves; but it's also the dividing line between an apple and the *idea* of an apple.

Why is embodiment relevant to these sorts of interactions with computers? It is relevant in at least three ways.

First, the designers of interactive systems have increasingly come to understand that interaction is intimately connected with the settings in which it occurs. In adopting anthropological techniques as ways to uncover the details of work and develop requirements for interactive systems to support that work, we have begun to realize just how important a role is played by the environment in which the work takes place.[4] This is true of both physical environments and social or organizational ones. Physical environments are arranged so as to make certain kinds of activities easier (or more difficult), and in turn, those activities are tailored to the details of the environment in which they take place. The same thing happens at an organizational level; the nature of the organization in which the work takes place will affect the work itself and the ways it is done. The increasing sensitivity to settings leads naturally to a concern with how work and interaction are embodied within those settings, because that embodiment determines how it is that computation and the setting will fit together.

Second, this focus on settings reflects a more general turn to consider work activities and artifacts in concrete terms rather than abstract ones. Instead of developing abstract accounts of mythical users, HCI increasingly employs field studies and observational techniques to stage "encounters" with real users, in real settings, doing real work. These encounters are often very revealing, as they show that the ways the work gets done are not the ways that are listed in procedural manuals, or even in the accounts that the people themselves would tell you if you asked. Attention to detail, to specifics, and to actual cases, leads in turn to thinking about computation in similar terms. In particular, it leads to a concern with how interaction is manifest in the interface. Tangible computing reflects this concern by exploring the opportunities for us to manifest computation and interaction in radically new forms, while social computing seeks ways for interaction to manifest more than simply the programmer's abstract model of the task, but also the specifics of how the work comes to be done. In the real world, where the artifacts through which interaction is conducted are directly embodied in the everyday environment, these are all manifested alongside each other, inseparably. Tangible and social computing are trying to stitch them back together after traditional interactive system design approaches ripped them apart.

Third, there is a recognition that, through their direct embodiment in the world we occupy, the artifacts of daily interaction can play many different roles. As an example, consider the revealing studies of the role of medical record cards in hospitals (Nygren, Johnson, and Henriksson 1992). From a technical perspective, patient record cards are simply carriers of well-defined information concerning the patient's diagnosis and treatment, and, as embodied on paper, present various problems: they can be lost, they can be hard to read, and they can only be in one place at a time. From this perspective, it seems both straightforward and beneficial to replace the paper records with electronic versions. However, in practice, such straightforward replacements are rarely successful. Studies of the failure of such systems show that the paper records are more than simply carriers of information about patients. They carry other important information as a result of the way that they are used in the work of the hospital. For example, handwriting on the forms reveals who performed different parts of the treatment; wear and tear on the form indicates heavy use; and the use of pencil marks rather than pen informally indicates tentative information. To trained eyes, a card conveys information not just about the patient, but also about the history of activities over the card and around the patient. It can do this because it not only represents the world of the patient, but it also participates in that world—it is an embodied artifact, and it participates in the embodied activities of those administering medical care. So, one relevance of embodiment for interaction with computational systems is that, for many tasks, it is relevant to consider how computation participates in the world it represents. Computation is fundamentally a representational medium, but as we attempt to expand the ways in which we interact with computation, we need to pay attention to the duality of representation and participation.

The third element of this book's argument is that the idea of embodiment as a common foundation points us to other schools of thought. Embodiment is not a new phenomenon, or a new area for intellectual

endeavor. In fact, it is a common theme running through much twentieth century thought. The notion of embodiment plays a special role in one particular school of philosophical thought, phenomenology.

Phenomenology is primarily concerned with how we perceive, experience, and act in the world around us. What differentiates it from other approaches is its central emphasis on the actual phenomena of experience, where other approaches might be concerned with abstract world models. Traditional approaches would suggest that we each have an understanding of the elements of which our world is constructed, and an abstract mental model of how these concepts are related. We understand that there are entities we can drink from, and that cups, glasses, and mugs are examples; we understand that we can sit on things like sofas and stools, and that people might keep cats and rabbits as house-pets, but rarely elephants or seals. This information, abstractly encoded in our heads, guides our actions in the world. Armed with a model of appropriate concepts and relations—an ontology—we can look around us and recognize what we see. So, the traditional model supposes that when I encounter a glass of wine, even though I have never seen this particular one before, I can still recognize it as being a glass of wine because of the way in which it fits into my model as an instance of the abstract class of glasses and other drinking vessels.

In contrast, the phenomenologists argue that the separation between mind and matter, or between what Descartes called the *res cogitans* and the *res extensa*, has no basis in reality. Thinking does not occur separately from being and acting. Certainly, there is nothing in our experience to support such a separation. In every case, we encounter them together, as aspects of the same existence. Consequently, phenomenology has attempted to reconstruct the relationship between experience and action without this separation. Rather than the Cartesians' theory- or model-driven approach to perception, the phenomenological approach argues for what we might call a *preontological* apprehension of the world. Perception begins with what is experienced, rather than beginning with what is expected; the model is to "see and understand" rather than "understand and see."

To say that phenomenology is all about perception is to limit it unfairly. In addition to perception, it is also concerned with action, with understanding, and with how these are all related to each other, as part and parcel of our daily experience as participants in the world. In the hands of some, such as Alfred Schutz, phenomenology has also been a tool to understand social action and practice; others such as Wittgenstein, while not phenomenologists, have developed allied approaches to topics such as language and meaning. As we will see, these approaches provide an extensive set of investigations of the questions of presence, embodiment, and action.

In turn, the fourth element of the book's argument is that we can build on the phenomenological understandings to create a foundational approach to embodied interaction. Such a foundation should do two things. First, it should account for the ways in which social and tangible computing—and, perhaps, further areas to be defined—are related to each other, showing how they can be draw upon each other's work and provide a unified model for Human-Computer Interaction. Second, it should inform and support the design, analysis and evaluation of interactive systems, providing us with ways of understanding how they work, from the perspective of embodiment.[5]

This, then, is the four-part hypothesis that this book sets out to explore: that tangible and social computing have a common basis; that embodiment is the core element they have in common; that embodiment is not a new idea, but has been a primary topic for phenomenology; and that phenomenology and related investigations of embodiment can provide material for developing a foundation for embodied interaction.

This has all been presented so far in very broad strokes. The chapters to come will explore the issues in more depth and provide much more background. The two chapters that follow describe the recent trends in HCI research that are the starting point for this work. Chapter 2 deals with tangible computing, while chapter 3 explores social computing. Each presents both the research and the context in which it emerged. However, they present tangible and social computing as self-contained; in chapter 4, we begin to examine how they might be brought together, and how ideas from phenomenology and other philosophies of presence and experience can be brought to bear to understand the relationships between them. Just as chapters 2 and 3 try to introduce the set of ideas from tangible and social computing that will inform the later discussion,

so chapter 4 provides an introduction to the phenomenological work that we will draw upon later. With this background, chapter 5 explores the notion of embodiment in more depth, drawing out a number of constituent elements whose relationships can be used to analyse interaction case studies. Chapter 6 builds on this and presents a framework that arranges these foundational elements to be able to draw on them for design, and chapter 7 points to some future directions.