Chapter 2

Understanding and conceptualizing interaction

2.1 Introduction

Imagine you have been asked to design an application to let people organize, store, and retrieve their email in a fast, efficient and enjoyable way. What would you do? How would you start? Would you begin by sketching out how the interface might look, work out how the system architecture will be structured, or even just start coding? Alternatively, would you start by asking users about their current experiences of saving email, look at existing email tools and, based on this, begin thinking about why, what, and how you were going to design the application?

Interaction designers would begin by doing the latter. It is important to realize that having a clear understanding of what, why, and how you are going to design something, before writing any code, can save enormous amounts of time and effort later on in the design process. Ill-thought-out ideas, incompatible and unusable designs can be ironed out while it is relatively easy and painless to do. Once ideas are committed to code (which typically takes considerable effort, time, and money), they become much harder to throw away—and much more painful. Such preliminary thinking through of ideas about user needs[1] and what

[1] User needs here are the range of possible requirements, including user wants and experiences.

kinds of designs might be appropriate is, however, a skill that needs to be learned. It is not something that can be done overnight through following a checklist, but requires practice in learning to identify, understand, and examine the issues—just like learning to write an essay or to program. In this chapter we describe what is involved. In particular, we focus on what it takes to understand and conceptualize interaction.

The main aims of this chapter are to:

- Explain what is meant by the problem space.
- Explain how to conceptualize interaction.
- Describe what a conceptual model is and explain the different kinds.
- Discuss the pros and cons of using interface metaphors as conceptual models.
- Debate the pros and cons of using realism versus abstraction at the interface.
- Outline the relationship between conceptual design and physical design.

2.2 Understanding the problem space

In the process of creating an interactive product, it can be temping to begin at the "nuts and bolts" level of the design. By this, we mean working out how to design the physical interface and what interaction styles to use (e.g., whether to use menus, forms, speech, icons, or commands). A problem with trying to solve a design problem beginning at this level is that critical usability goals and user needs may be overlooked. For example, consider the problem of providing drivers with better navigation and traffic information. How might you achieve this? One could tackle the problem by thinking straight away about a good technology or kind of interface to use. For example, one might think that augmented reality, where images are superimposed on objects in the real world (see Figure 2.1 on Color Plate 2), would be appropriate, since it can be useful for integrating additional information with an ongoing activity (e.g., overlaying X-rays on a patient during an operation). In the context of driving, it could be effective for displaying information to drivers who need to find out where they are going and what to do at certain points during their journey. In particular, images of places and directions to follow could be projected inside the car, on the dashboard or rear-view mirror. However, there is a major problem with this proposal: it is likely to be very unsafe. It could easily distract drivers, luring them to switch their attention from the road to where the images were being projected.

A problem in starting to solve a design problem at the physical level, therefore, is that usability goals can be easily overlooked. While it is certainly necessary at some point to decide on the design of physical aspects, it is better to make these kinds of design decisions *after* understanding the nature of the problem space. By this, we mean conceptualizing what you want to create and articulating why you want to do so. This requires thinking through how your design will support people in their everyday or work activities. In particular, you need to ask yourself whether the interactive product you have in mind will achieve what you hope it will. If so,

how? In the above example, this involves finding out what is problematic with existing forms of navigating while driving (e.g., trying to read maps while moving the steering wheel) and how to ensure that drivers can continue to drive safely without being distracted.

Clarifying your usability and user experience goals is a central part of working out the problem space. This involves making explicit your implicit assumptions and claims. Assumptions that are found to be vague can highlight design ideas that need to be better formulated. The process of going through them can also help to determine relevant user needs for a given activity. In many situations, this involves identifying human activities and interactivities that are problematic and working out how they might be improved through being supported with a different form of interaction. In other situations it can be more speculative, requiring thinking through why a novel and innovative use of a new technology will be potentially useful.

Below is another scenario in which the problem space focuses on solving an identified problem with an existing product. Initial assumptions are presented first, followed by a further explanation of what lies behind these (assumptions are highlighted in italics):

> A large software company has decided to develop an upgrade of its web browser. *They assume that there is a need for a new one, which has better and more powerful functionality.* They begin by carrying out an extensive study of people's actual use of web browsers, talking to lots of different kinds of users and observing them using their browsers. One of their main findings is that many people do not use the bookmarking feature effectively. A common finding is that it is too restrictive and underused. *In fathoming why this is the case, it was considered that the process of placing web addresses into hierarchical folders was an inadequate way of supporting the user activity* of needing to mark hundreds and sometimes thousands of websites such that any one of them could be easily returned to or forwarded onto other people. *An implication of the study was that a new way of saving and retrieving web addresses was needed.*

In working out why users find the existing feature of bookmarking cumbersome to use, a further assumption was explicated:

- *The existing way of organizing saved (favorite) web addresses into folders is inefficient because it takes too long and is prone to errors.*

A number of underlying reasons why this was assumed to be the case were further identified, including:

- It is easy to lose web addresses by placing them accidentally into the wrong folders.
- It is not easy to move web addresses between folders.
- It is not obvious how to move a number of addresses from the saved favorite list into another folder simultaneously.
- It is not obvious how to reorder web addresses once placed in folders.

Based on this analysis, a set of assumptions about the user needs for supporting this activity more effectively were then made. These included:

- ***If the bookmarking function was improved users would find it more useful and use it more to organize their web addresses.***
- ***Users need a flexible way of organizing web addresses they want to keep for further reference or for sending on to other people.***

A framework for explicating assumptions

Reasoning through your assumptions about why something might be a good idea enables you to see the strengths and weaknesses of your proposed design. In so doing, it enables you to be in a better position to commence the design process. We have shown you how to begin this, through operationalizing relevant usability goals. In addition, the following questions provide a useful framework with which to begin thinking through the problem space:

- Are there problems with an existing product? If so, what are they? Why do you think there are problems?
- Why do you think your proposed ideas might be useful? How do you envision people integrating your proposed design with how they currently do things in their everyday or working lives?
- How will your proposed design support people in their activities? In what way does it address an identified problem or extend current ways of doing things? Will it really help?

ACTIVITY 2.1 At the turn of the millennium, WAP-enabled (wireless application protocol) phones came into being, that enabled people to connect to the Internet using them. To begin with, the web-enabled services provided were very primitive, being text-based with limited graphics capabilities. Access was very restricted, with the downloaded information being displayed on a very small LCD screen (see Figure 2.2). Despite this major usability drawback, every telecommunication company saw this technological breakthrough as an opportunity to create innovative applications. A host of new services were explored, including text messaging, online booking of tickets, betting, shopping, viewing movies, stocks and shares, sports events and banking.

What assumptions were made about the proposed services? How reasonable are these assumptions?

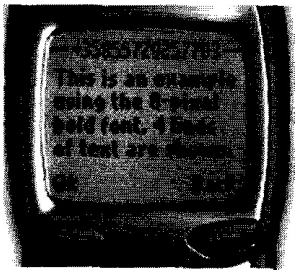


Figure 2.2 An early cell phone display. Text is restricted to three or four lines at a time and scrolls line by line, making reading very cumbersome. Imagine trying to read a page from this book in this way! The newer 3G (third generation) phones have bigger displays, more akin to those provided with handheld computers.

Comment

The problem space for this scenario was very *open-ended.* There was no identifiable problem that needed to be improved or fixed. Alternatively, the new WAP technology provided opportunities to create new facilities and experiences for people. One of the main assumptions is that *people want to be kept informed* of up-to-the-minute news (e.g. sports, stocks and share prices) *wherever they are.* Other assumptions included:

- That people want to be able to decide what to do in an evening while on their way home from work (e.g., checking TV listings, movies, making restaurant reservations).
- That people *want to be able to interact with information on the move* (e.g., reading email on the train).
- That users are prepared to put up with a very small display and will be happy browsing and interacting with information using a restricted set of commands via a small number of tiny buttons.
- That people *will be happy* doing things on a mobile phone that they normally do using their PCs (e.g., reading email, surfing the web, playing video games, doing their shopping).

It is reasonable to assume that people want flexibility. They like to be able to find out about news and events wherever they are (just look at the number of people who take a radio with them to a soccer match to find out the scores of other matches being played at the same time). People also like to use their time productively when traveling, as in making phone calls. Thus it is reasonable to assume they would like to read and send email on the move. The most troublesome assumption is whether people are prepared to interact with the range of services proposed using such a restricted mode of interactivity. In particular, it is questionable whether most people are prepared to give up what they have been used to (e.g. large screen estate, ability to type messages using a normal-sized keyboard) for the flexibility of having access to very restricted Internet-based information via a cell phone they can keep in their pocket.

One of the benefits of working through your assumptions for a problem space before building anything is that it can highlight problematic concerns. In so doing, it can identify ideas that need to be reworked, before it becomes too late in the design process to make changes. Having a good understanding of the problem space can also help greatly in formulating what it is you want to design. Another key aspect of conceptualizing the problem space is to think about the overall structure of what will be built and how this will be conveyed to the users. In particular, this involves developing a conceptual model.

2.3 Conceptual models

> *"The most important thing to design is the user's conceptual model. Everything else should be subordinated to making that model clear, obvious, and substantial. That is almost exactly the opposite of how most software is designed." (David Liddle, 1996, p. 17)*

By a conceptual model is meant:

> ***a description of the proposed system in terms of a set of integrated ideas and concepts about what it should do, behave and look like, that will be understandable by the users in the manner intended.***

To develop a conceptual model involves envisioning the proposed product, based on the users' needs and other requirements identified. To ensure that it is designed to be understandable in the manner intended requires doing iterative testing of the product as it is developed. A key aspect of this design process is initially to decide what the users will be doing when carrying out their tasks. For example, will they be primarily searching for information, creating documents, communicating with other users, recording events, or some other activity? At this stage, the interaction mode that would best support this needs to be considered. For example, would allowing the users to browse be appropriate, or would allowing them to ask questions directly to the system in their native language be more effective? Decisions about which kind of interaction style to use (e.g., whether to use a menu-based system, speech input, commands) should be made in relation to the interaction mode. Thus, decisions about which mode of interaction to support differ from those made about which style of interaction to have; the former being at a higher level of abstraction. The former are also concerned with determining the nature of the users' activities to support, while the latter are concerned with the selection of specific kinds of interface.

Once a set of possible ways of interacting with an interactive system has been identified, the design of the conceptual model then needs to be thought through in terms of actual concrete solutions. This entails working out the behavior of the interface, the particular interaction styles that will be used, and the "look and feel" of the interface. At this stage of "fleshing out," it is always a good idea to explore a number of possible designs and to assess the merits and problems of each one.

Another way of designing an appropriate conceptual model is to select an interface metaphor. This can provide a basic structure for the conceptual model that is couched in knowledge users are familiar with. Examples of well-known interface metaphors are the desktop and search engines (which we will cover in Section 2.4). Interaction paradigms can also be used to guide the formation of an appropriate conceptual metaphor. They provide particular ways of thinking about interaction design, such as designing for desktop applications or ubiquitous computing (these will also be covered in Section *2.5).*

As with any aspect of interaction design, the process of fleshing out conceptual models should be done iteratively, using a number of methods. These include sketching out ideas, storyboarding, describing possible scenarios, and prototyping aspects of the proposed behavior of the system. All these methods will be covered in Chapter 8, which focuses on *doing* conceptual design. Here, we describe the different kinds of conceptual models, interface metaphors, and interaction paradigms to give you a good understanding of the various types prior to thinking about how to design them.

There are a number of different kinds of conceptual models. These can be broken down into two main categories: those based on activities and those based on objects.

2.3.1 Conceptual models based on activities

The most common types of activities that users are likely to be engaged in when interacting with systems are:

1. instructing
2. conversing
3. manipulating and navigating
4. exploring and browsing

A first thing to note is that the various kinds of activity are not mutually exclusive, as they can be carried out together. For example, it is possible for someone to give instructions while conversing or navigate an environment while browsing. However, each has different properties and suggests different ways of being developed at the interface. The first one is based on the idea of letting the user issue instructions to the system when performing tasks. This can be done in various interaction styles: typing in commands, selecting options from menus in a windows environment or on a touch screen, speaking aloud commands, pressing buttons, or using a combination of function keys. The second one is based on the user conversing with the system as though talking to someone else. Users speak to the system or type in questions to which the system replies via text or speech output. The third type is based on allowing users to manipulate and navigate their way through an environment of virtual objects. It assumes that the virtual environment shares some of the properties of the physical world, allowing users to use their knowledge of how physical objects behave when interacting with virtual objects. The fourth kind is based on the system providing information that is structured in such a way as to allow users to find out or learn things, without having to formulate specific questions to the system.

ACTIVITY 2.2 A company is building a wireless information system to help tourists find their way around an unfamiliar city. What would they need to find out in order to develop a conceptual model?

Comment To begin, they would need to ask: what do tourists want? Typically, they want to find out lots of things, such as how to get from A to B, where the post office is and where a good Chinese restaurant is. They then need to consider how best to support the activity of requesting information. Is it preferable to enable the tourists to ask questions of the system as if they were having a conversation with another human being? Or would it be more appropriate to allow them to ask questions as if giving instructions to a machine? Alternatively, would they prefer a system that structures information in the form of lists, maps, and recommendations that they could then explore at their leisure?

1. Instructing

This kind of conceptual model describes how users carry out their tasks through instructing the system what to do. Examples include giving instructions to a system to perform operations like tell the time, print a file, and remind the user of an appointment. A diverse range of devices has been designed based on this model, including VCRs, hi-fi systems, alarm clocks, and computers. The way in which the user issues instructions can vary from pressing buttons to typing in strings of characters. Many activities are readily supported by giving instructions.

Operating systems like Unix and DOS have been specifically designed as command-based systems, to which the user issues instructions at the prompt as a command or set of commands. In Windows and other GUI-based systems, control keys or the selection of menu options via a mouse are used. Well-known applications that are command-based include word processing, email, and CAD. Typically, a wide range of functions is provided from which users choose when they want to do something to the object they are working on. For example, a user writing a report using a word processor will want to format the document, count the numbers of words typed, and check the spelling. The user will need to instruct the system to do these operations by issuing apprbpriate commands. Typically, commands are carried out in a sequence, with the system responding appropriately (or not) as instructed.

One of the main benefits of an instruction-based conceptual model is that it supports quick and efficient interaction. It is particularly suited to repetitive kinds of actions performed on multiple objects. Examples include the repetitive actions of saving, deleting, and organizing email messages or files.

ACTIVITY 2.3 There are many different kinds of vending machines in the world. Each offers a range of goods, requiring the user initially to part with some money. Figure 2.3 shows photos of two different vending machines, one that provides soft drinks and the other a range of snacks. Both support the interaction style of issuing instructions. However, the way they do it is quite different.

What instructions must be issued to obtain a can of soft drink from the first machine and a bar of chocolate from the second? Why has it been necessary to design a more complex mode of interaction for the second vending machine? What problems can arise with this mode of interaction?

Comment The first vending machine has been designed on a very simple instruction-based conceptual model. There are a small number of drinks to choose from and each is represented by a large button displaying the label of each drink. The user simply has to press one button and (hopefully) this will have the effect of returning the selected drink. The second machine is more complex, offering a wider range of snacks. The trade-off for providing more choices, however, is that the user can no longer instruct the machine by using a simple one-press action but is required to use a more complex process, involving: (i) reading off the code (e.g., C12) under the item chosen, then (ii) keying this into the number pad adjacent to the displayed items, and (iii) checking the price of the selected option and ensuring that the amount of money inserted is the same or more (depending on whether or not the machine provides change). Problems that can arise from this mode of interaction are the customer

Figure 2.3 Two vending machines, (a) one selling soft drinks, (b) the other selling a range of snacks.

misreading the code and or mistyping in the code, resulting in the machine not issuing the snack or providing the wrong sort.

A better way of designing an interface for a large number of choices of variable cost is to continue to use direct **mapping,** but use buttons that show miniature versions of the snacks placed in a large matrix (rather than showing actual versions). This would use the available space at the front of the vending machine more economically. The customer would need only to press the button of the object chosen and put in the correct amount of money.

Much research has been carried out on how to optimize command-based and other instruction-giving systems with respect to usabilty goals. The form of the commands (e.g., the use of abbreviations, full names, icons, and/or labels), their syntax (how best to combine different commands), and their organization (e.g., how to structure options in different menus) are examples of some of the main areas that have been investigated (Shneiderman, 1998). In addition, various cognitive issues have been investigated that we will look at in the next chapter, such as the problems people have in remembering the names of a set of commands. Less

research has been carried out, however, on the best way to design the ordering and sequencing of button pressing for physical devices like cell phones, calculators, remote controls and vending machines.

ACTIVITY 2.4 Another ubiquitous vending machine is the ticket machine. Typically, a number of instructions have to be given in a sequence when using one of these. Consider ticket machines designed to issue train tickets at railway stations—how often have you (or the person in front of you) struggled to work out how to purchase a ticket and made a mistake? How many instructions have to be given? What order are they given in? Is it logical or arbitrary? Could the interaction have been designed any differently to make it more obvious to people how to issue instructions to the machine to get the desired train ticket?

Comment Ticketing machines vary enormously from country to country and from application to application. There seems to be little attempt to standardize. Therefore, a person's knowledge of the Eurostar ticketing machine will not be very useful when buying a ticket for the Sydney Monorail or cinema tickets for the Odeon. Sometimes the interaction has been designed to get you to specify the type of ticket first (e.g. adult, child), the kind of ticket (e.g. single, return, special saver), then the destination, and finally to insert their money. Others require that the user insert a credit card first, before selecting the destination and the type of ticket.

2. Conversing

This conceptual model is based on the idea of a person conversing with a system, where the system acts as a dialog partner. In particular, the system is designed to respond in a way another human being might when having a conversation with someone else. It differs from the previous category of instructing in being intended to reflect a more two-way communication process, where the system acts more like a partner than a machine that simply obeys orders. This kind of conceptual model has been found to be most useful for applications in which the user needs to find out specific kinds of information or wants to discuss issues. Examples include advisory systems, help facilities, and search engines. The proposed tourist application described earlier would fit into this category.

The kinds of conversation that are supported range from simple voice-recognition menu-driven systems that are interacted with via phones to more complex natural-language-based systems that involve the system parsing and responding to user queries typed in by the user. Examples of the former include banking, ticket booking, and train time inquiries, where the user talks to the system in single-word phrases (e.g., yes, no, three) in response to prompts from the system. Examples of the latter include search engines and help systems, where the user types in a specific query (e.g., how do I change the margin widths?) to which the system responds by giving various answers.

A main benefit of a conceptual model based on holding a conversation is that it allows people, especially novices, to interact with a system in a way they are already familiar with. For example, the search engine "Ask Jeeves for Kids!" allows children to ask a question in a way they would when asking their teachers or parents—rather than making them reformulate their question in terms of key words and Boolean logic. A disadvantage of this approach, however, is the misunderstandings that can arise when the search engine is unable to answer the child's question in the

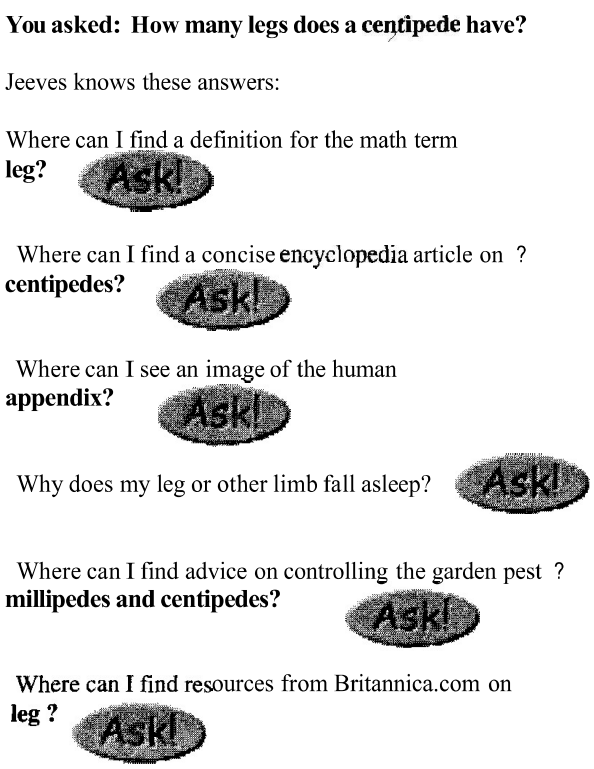
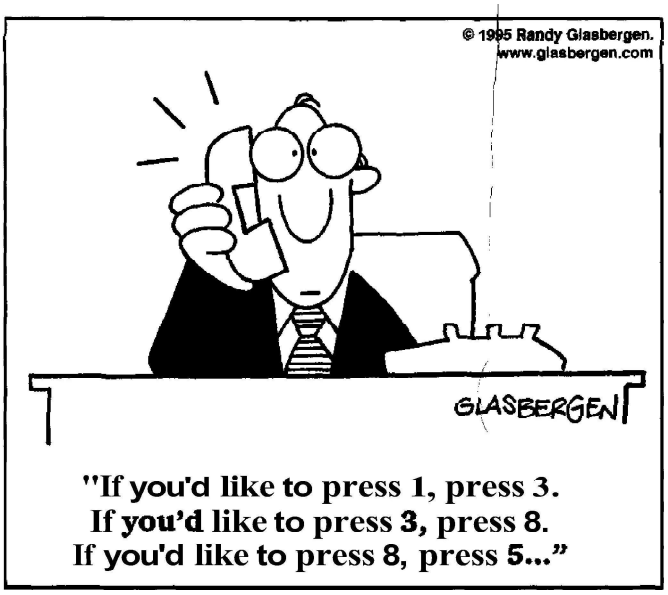


Figure 2.4 The response from "Ask Jeeves for Kids!" search engine when asked "how many legs does a centipede have?"

way the child expects. For example, a child might type in a seemingly simple question, like "How many legs does a centipede have?" which the search engine finds difficult to answer. Instead, the search engine replies by suggesting a number of possible websites that may be relevant but–as can be seen in Figure 2.4–can be off the mark.

Another problem that can arise from a conversational-based, conceptual model is that certain kinds of tasks are transformed into cumbersome and one-sided interactions. This is especially the case for automated phone-based systems that use auditory menus to advance the conversation. Users have to listen to a voice providing several options, then make a selection, and repeat through further layers of menus before accomplishing their goal (e.g., reaching a real human, paying a bill). Here is the beginning of a dialog between a user who wants to find out about car insurance and an insurance company's reception system:

```
<user dials an insurance company>
"Welcome to St. Paul's Insurance Company. Press 1 if new
customer, 2 if you are an existing customer".
<user presses 1>
"Thank you for calling St. Paul's Insurance Company. If you
require house insurance press 1, car insurance press 2,
travel insurance press 3, health insurance press 4, other
press 5"
<user presses 2>
"You have reached the car insurance division. If you re-
quire information about fully comprehensive insurance press
1, 3rd-party insurance press 2..."
```



A recent development based on the conversing conceptual model is animated agents. Various kinds of characters, ranging from "real" people appearing at the interface (e.g., videoed personal assistants and guides) to cartoon characters (e.g., virtual and imaginary creatures), have been designed to act as the partners in the conversation with the system. In so doing, the dialog partner has become highly visible and tangible, appearing to both act and talk like a human being (or creature). The user is able to see, hear, and even touch the partner (when it is a physical toy) they are talking with, whereas with other systems based on a dialog partner (e.g., help systems) they can only hear or read what the system is saying. Many agents have also been designed to exhibit desirable human-like qualities (e.g., humorous, happy, enthusiastic, pleasant, gentle) that are conveyed through facial expressions and lifelike physical movements (head and lip movements, body movements). Others have been designed more in line with Disney-like cartoon characters, exhibiting exaggerated behaviors (funny voices, larger-than-life facial expressions).

Animated agents that exhibit human-like or creature-like physical behavior as well as "talk" can be more believable. The underlying conceptual model is conveyed much more explicitly through having the system act and talk via a visible agent. An advantage is that it can make it easier for people to work out that the interface agent (or physical toy) they are conversing with is not a human being, but a synthetic character that has been given certain human qualities. In contrast, when the dialog partner is hidden from view, it is more difficult to discern what is behind it and just how intelligent it is. The lack of visible cues can lead users into thinking it is more intelligent than it actually is. If the dialog partner then fails to understand their questions or comments, users are likely to lose patience with it. Moreover,

they are likely to be less forgiving of it (having been fooled into thinking the dialog partner is more intelligent than it really is) than of a dialog partner that is represented as a cartoon character at the interface (having only assumed it was a simple partner). The flip side of imbuing dialog partners with a physical presence at the interface, however, is that they can turn out to be rather annoying (for more on this topic see Chapter 5).

3. Manipulating and navigating

This conceptual model describes the activity of manipulating objects and navigating through virtual spaces by exploiting users' knowledge of how they do this in the physical world. For example, virtual objects can be manipulated by moving, selecting, opening, closing, and zooming in and out of them. Extensions to these actions can also be included, such as manipulating objects or navigating through virtual spaces, in ways not possible in the real world. For example, some virtual worlds have been designed to allow users to teleport from place to place or to transform one object into another.

A well known instantidtion of this kind of conceptual model is direct manipulation. According to Ben Shneiderman (1983), who coined the term, direct-manipulation interfaces possess three fundamental properties:

- continuous representation of the objects and actions of interest
- rapid reversible incremental actions with immediate feedback about the object of interest
- physical actions and button pressing instead of issuing commands with complex syntax

Benefits of direct manipulation interfaces include:

- helps beginners learn basic functionality rapidly
- experienced users can work rapidly on a wide range of tasks
- infrequent users can remember how to carry out operations over time
- no need for error messages, except very rarely

  users can immediately see if their actions are furthering their goals and if not do something else
- users experience less anxiety
- users gain confidence and mastery and feel in control

Apple Computer Inc. was one of the first computer companies to design an operating environment using direct manipulation as its central mode of interaction. The highly successful Macintosh desktop demonstrates the main principles of direct manipulation (see Figure 2.5). To capitalize on people's understanding of what happens to physical objects in the real world, they used a number of visual and auditory cues at the interface that were intended to emulate them. One of

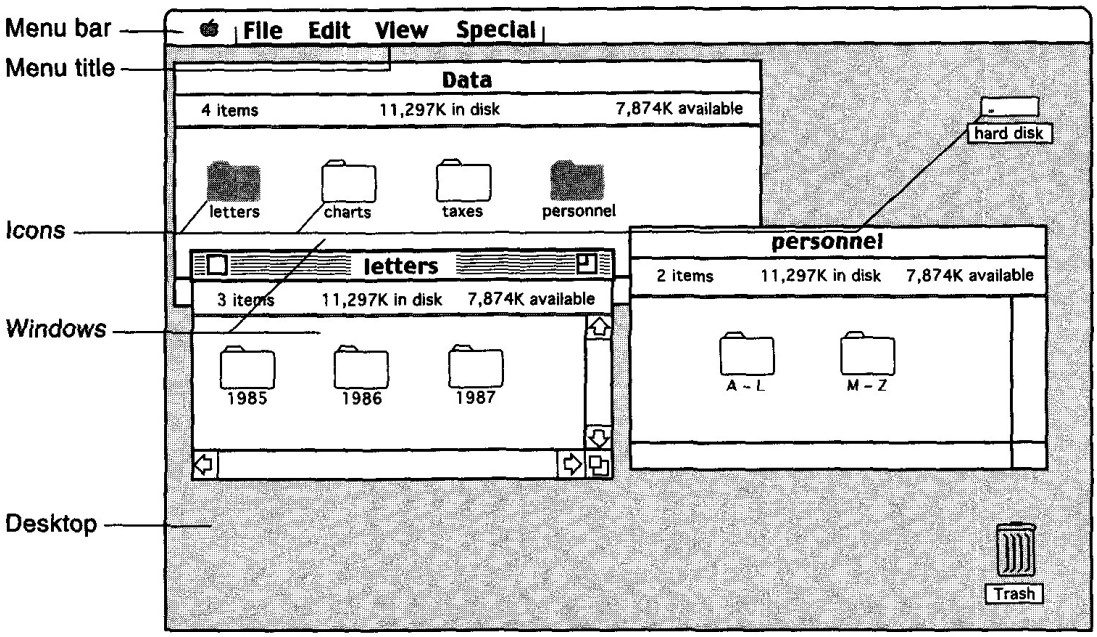


Figure 2.5 Original Macintosh desktop interface.

their assumptions was that people expect their physical actions to have physical results, so when a drawing tool is used, a corresponding line should appear and when a file is placed in the trash can a corresponding sound or visual cue showing it has been successfully thrown away is used (Apple Computer Inc., 1987). A number of specific visual and auditory cues were used to provide such feedback, including various animations and sounds (e.g. shrinking and expanding icons accompanied with 'shhhlicc' and '*crouik*' sounds to represent opening and closing of files). Much of this interaction design was geared towards providing clues to the user to know what to do, to feel comfortable, and to enjoy exploring the interface.

Many other kinds of direct manipulation interfaces have been developed, including video games, data visualization tools and CAD systems. Virtual environments and virtual reality have similarly employed a range of interaction mechanisms that enable users to interact with and navigate through a simulated 3D physical world. For example, users can move around and explore aspects of a 3D environment (e.g., the interior of a building) while also moving objects around in the virtual environment, (e.g., rearranging the furniture in a simulated living room). Figure 2.6 on Color Plate 3 shows screen shots of some of these.

While direct manipulation and virtual environments provide a very versatile mode of interaction, they do have a number of drawbacks. At a conceptual level, some people may take the underlying conceptual model too literally and expect certain things to happen at the interface in the way they would in the physical world. A well known example of this phenomenon is of new Mac users being terri-

fied of dragging the icon of their floppy disk to the trash can icon on the desktop to eject it from the computer for fear of deleting it in the same way files are when placed in the trash can. The conceptual confusion arises because the designers opted to use the same action (dropping) on the same object (trash can) for two completely different operations, deleting and ejecting. Another problem is that not all tasks can be described by objects and not all actions can be done directly. Some tasks are better achieved through issuing instructions and having textual descriptions rather than iconic representations. Imagine if email messages were represented as small icons in your mailbox with abbreviations of who they were from and when they were sent. Moreover, you could only move them around by dragging them with a mouse. Very quickly they would take up your desk space and you would find it impossible to keep track of them all.

4. Exploring and browsing

This conceptual model is based on the idea of allowing people to explore and browse information, exploiting their knowledge of how they do this with existing media (e.g., books, magazines, TV, radio, libraries, pamphlets, brochures). When people go to a tourist office, a bookstore, or a dentist's surgery, often they scan and flick through parts of the information displayed, hoping to find something interesting to read. CD-ROMs, web pages, portals and e-commerce sites are applications based on this kind of conceptual model. Much thought needs to go into structuring the information in ways that will support effective navigation, allowing people to search, browse, and find different kinds of information.

ACTIVITY 2.5 What conceptual models are the following applications based on?

(a) a 3D video game, say a car-racing game with a steering wheel and tactile, audio, and visual feedback

(b) the Windows environment

(c) a web browser

Commenf

(a) A 3D video game is based on a direct manipulation/virtual environment conceptual model.

(b) The Windows environment is based on a hybrid form of conceptual model. It combines a manipulating mode of interaction where users interact with menus, scrollbars, documents, and icons, an instructing mode of interaction where users can issue commands through selecting menu options and combining various function keys, and a conversational model of interaction where agents (e.g. Clippy) are used to guide users in their actions.

(c) A web browser is also based on a hybrid form of conceptual model, allowing users to explore and browse information via hyperlinks and also to instruct the network what to search for and what results to present and save.

BOX 2.1 Which is Best—Agents, Direct Manipulations, or Commands?

An ongoing debate in interaction design concerns the pros and cons of using direct manipulation versus interface agents. Nicholas Negroponte (MIT Media Lab), a strong advocate of the agents approach, claims that they can be much more versatile than direct manipulation interfaces, allowing users to do what they want to do through delegating the boring and time-consuming tasks to an agent. He describes the analogy of a well-trained English butler, who answers the phone, tends to a person's needs, fends off callers, and tells 'white lies' if necessary on his master's behalf. Similarly, a *digital* butler is designed to read a user's email and flag the important ones, scout the web and newsgroups for interesting information, screen unwanted electronic intrusions, and so on. His vision is based on the assumption that people like to delegate work to others rather than directly manipulating computers themselves.

In opposition, Ben Shneiderman (University of Maryland) warns of the dangers of delegating tasks to agents, pointing out how difficult it is to train an agent to do all the things users want done in the way they want them done. If the agents do the tasks incorrectly or do not understand what the user wants, frustration and anger will ensue. Moreover, he argues that users do not want to be constantly monitored and told what to do by the computer. Consider the analogy of your car deciding you should be driving more slowly because it is raining. He suggests that direct manipulation has many more advantages, allowing users to enjoy mastery and being in control. He points out how people like to know what is going on, be involved in the action and have a sense of power over the computer—all of which direct manipulation interfaces support.

Another perspective on this debate is that in fact many tasks are best carried out at an abstract level, involving neither manipulation nor conversing with an agent. Issuing abstract commands based on a carefully designed set of syntax and semantics is often a very efficient and elegant way of performing operations. This is especially the case for repetitive operations, where often the same action needs to be performed on multiple objects. Examples include sorting out files, deleting accumulated email messages, opening and closing files, and installing applications comprising multiple files—which when done by direct manipulation or through delegation can be inefficient or ambiguous.

Consider how you would edit an essay using a word processor. Suppose you had referenced work by Ben Shneiderman but had spelled his name as Schneiderman, with an extra "c" throughout the essay. How could you correct this error using a direct manipulation interface? You would need to read through your essay and manually select the "c" in every "Schneiderman," highlighting and then deleting it. This is tedious and it would be easy to miss one or two. By contrast, this operation would be relatively effortless and also likely to be more accurate by issuing commands. You would simply need to instruct the word processor to *find* every "Schneiderman" and *replace* it with "Shneiderman." This could be done through either speaking the commands or typing them into a dialog box.

As a compromise, several interaction designers have recognized the need to support abstract classes of action in direct manipulation interfaces, while allowing for command-based and dialog modes of interaction in direct manipulation interfaces. However, as mentioned earlier, such redundancy can result in a more complex conceptual model that the user will have to spend more time learning.

ACTIVITY 2.6 Which conceptual model or combination of models do you think is most suited to supporting the following user activities?

(a) downloading music off the web

(b) programming

Comment

(a) The activity involves selecting, saving, cataloging and retrieving large files from an external source. Users need to be able to browse and listen to samples of the music and then instruct the machine to save and catalog the files in an order that they can readily access at subsequent times. A conceptual model based on instructing and navigating would seem appropriate.

(b) Programming involves various activities including checking, debugging, copying libraries, editing, testing, and annotating. An environment that supports this range of tasks needs to be flexible. A conceptual model that allows visualization and easy manipulation of code plus efficient instructing of the system on how to check, debug, copy, etc., is essential.

2.3.2 Conceptual models based on objects

The second category of conceptual models is based on an object or artifact, such as a tool, a book, or a vehicle. These tend to be more specific than conceptual models based on activities, focusing on the way a particular object is used in a particular context. They are often based on an analogy with something in the physical world. An example of a highly successful conceptual model based on an object is the spreadsheet (Winograd, 1996). The object this is based on is the ledger sheet.

The first spreadsheet was designed by Dan Bricklin, and called VisiCalc. It enabled people to carry out a range of tasks that previously could only be done very laboriously and with much difficulty using other software packages, a calculator, or by hand (see Figure 2.7). The main reasons why the spreadsheet has become so successful are first, that Bricklin ***understood*** what kind of tool would be useful to people in the financial world (like accountants) and second, he knew how to design it so that it could be used in the way that these people would find useful. Thus, at the outset, he understood (i) the kinds of activities involved in the financial side of business, and (ii) the problems people were having with existing tools when trying to achieve these activities.

A core financial activity is forecasting. This requires projecting financial results based on assumptions about a company, such as projected and actual sales, investments, infrastructure, and costs. The amount of profit or loss is calculated for different projections. For example, a company may want to determine how much loss it will incur before it will start making a profit, based on different amounts of investment, for different periods of time. Financial analysts need to see a spread of projections for different time periods. Doing this kind of multiple projecting by hand requires much effort and is subject to errors. Using a calculator can reduce the computational load of doing numerous sums, but it still requires the person to do much key pressing and writing down of partial results—again making the process vulnerable to errors.

To tackle these problems, Bricklin exploited the interactivity provided by microcomputers and developed an application that was capable of ***interactive*** financial

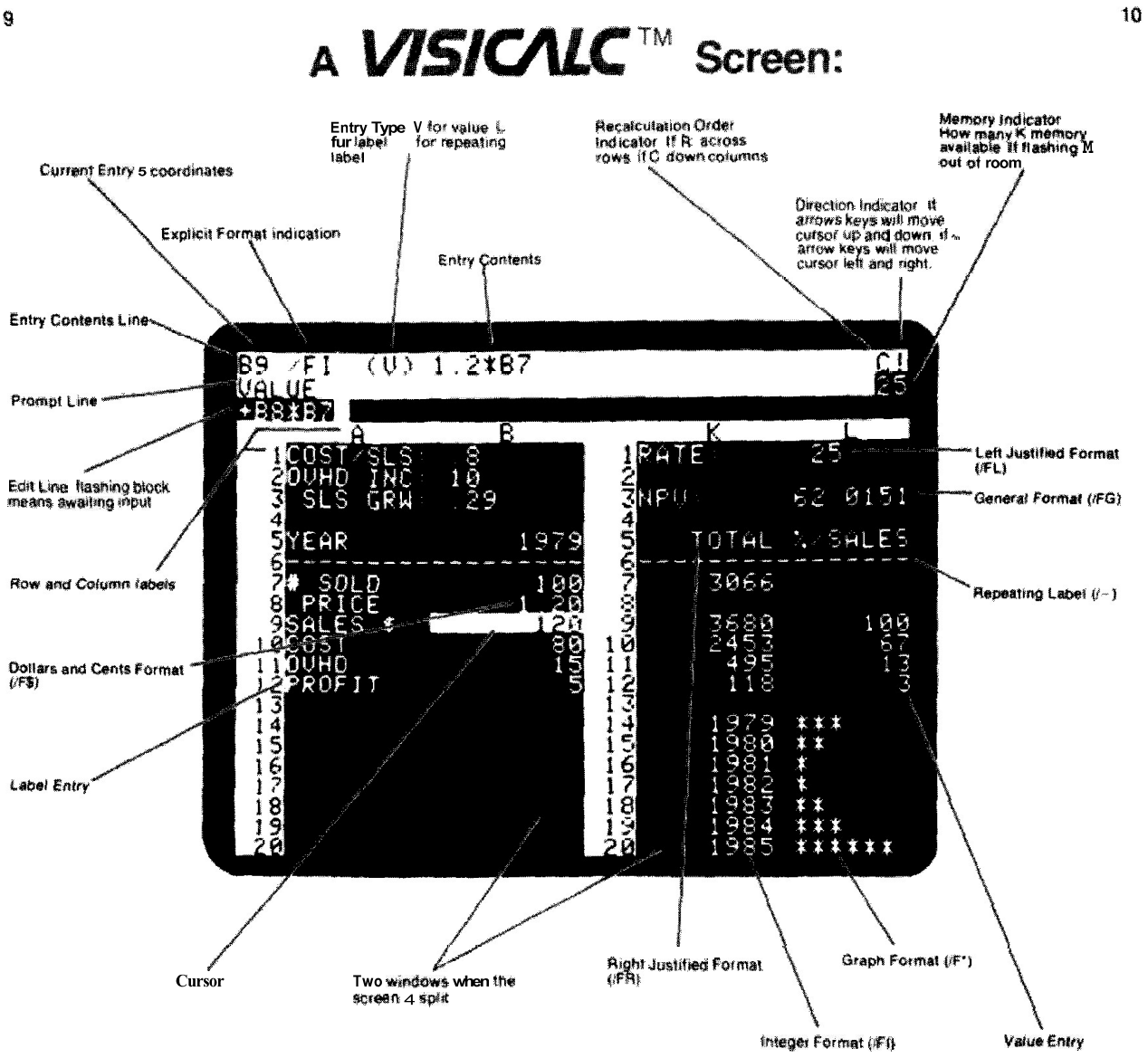


Figure 2.7 Reference card showing annotated screen dump for VisiCalc

modeling. Key aspects of his conceptual model were: (i) to create a spreadsheet that was *analogous* to a ledger sheet in the way it looked, with columns and rows, which allowed people to capitalize on their familiarity with how to use this kind of representation, (ii) to make the spreadsheet interactive, by allowing the user to input and change data in any of the cells in the columns or rows, and (iii) to get the computer to perform a range of different calculations and recalculations in response to user input. For example, the last column can be programmed to display the sum of all the cells in the columns preceding it. With the computer doing all the calculations, together with an easy-to-learn-and-use interface, users were provided with an ***easy-to-understand*** tool. Moreover, it gave them a new way of effortlessly working out any

number of forecasts—greatly extending what they could do before with existing tools.

Another popular accounting tool intended for the home market, based on a conceptual model of an object, is Quicken. This used paper checks and registers for its basic structure. Other examples of conceptual models based on objects include most operating environments (e.g., Windows and the Mac desktop) and web portals. All provide the user with a familiar frame of reference when starting the application.

BOX 2.2 The Star Interface *(Based On Miller and Johnson, 1996 and Smith et al., 1982)*

In 1981, Xerox introduced the 8010 "Star" system, which revolutionized the way interfaces were designed for personal computing. Although not commercially successful, many of the ideas behind its design were borrowed and adapted by other companies, such as the Apple Mac and Microsoft Windows, which then became highly successful.

Star was designed as an office system, targeted at workers not interested in computing *per se*. An important design goal, therefore, was to make the computer as "invisible" to the users as possible and to design applications that were suitable for them. The Star developers spent several person-years at the beginning of the project working out an appropriate conceptual model for such an office system. In the end they selected a conceptual model based on a physical office. They wanted the office workers to imagine the computer to be like an office environment, by acting on electronic counterparts of physical objects in the real world. Their assumption was that this would simplify and clarify the electronic world, making it seem more familiar, less alien and easier to learn, (see Figure 2.8).

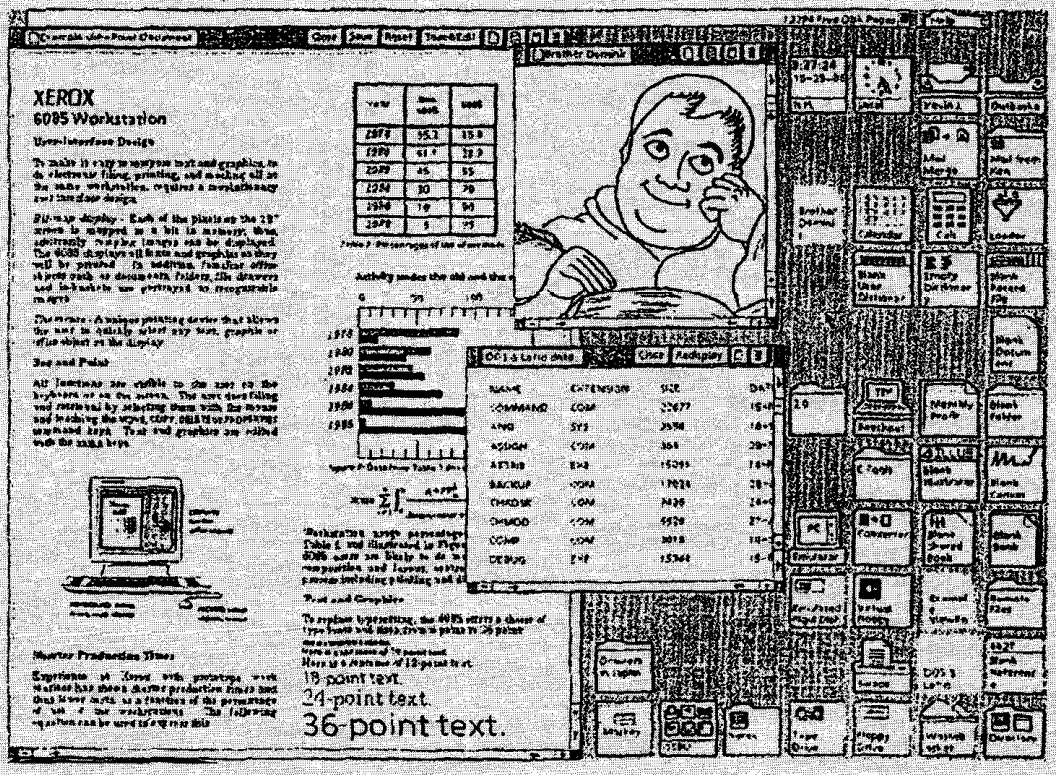


Figure 2.8 The Star interface.

2.3.3 A case of mix and match?

As we have pointed out, which kind of conceptual model is optimal for a given application obviously depends on the nature of the activity to be supported. Some are clearly suited to supporting a given activity (e.g., using manipulation and navigation for a flight simulator) while for others, it is less clear what might be best (e.g., writing and planning activities may be suited to both manipulation and giving instructions). In such situations, it is often the case that some form of hybrid conceptual model that combines different interaction styles is appropriate. For example, the tourist application in Activity 2.2 may end up being optimally designed based on a combination of conversing and exploring models. The user could ask specific questions by typing them in or alternatively browse through information. Shopping on the Internet is also often supported by a range of interaction modes. Sometimes the user may be browsing and navigating, other times communicating with an agent, at yet other times parting with credit card details via an instruction-based form fill-in. Hence, which mode of interaction is "active" depends on the stage of the activity that is being carried out.

BOX 2.3 Do Users Understand The Conceptual Model In The Way Intended?

A fundamental part of developing a conceptual model is to determine whether the ideas generated about how the system should look and behave will be understood by the users in the manner intended. Norman (1988) has provided a framework to elucidate the relationship between the design of a conceptual model and a user's understanding of it (see Figure 2.9). Essentially, there are three interacting components: the designer, the user, and the system. Behind each of these are three interlinking conceptual models:

- the design model—the model the designer has of how the system should work
- the system image—how the system actually works
- the user's model—how the user understands how the system works

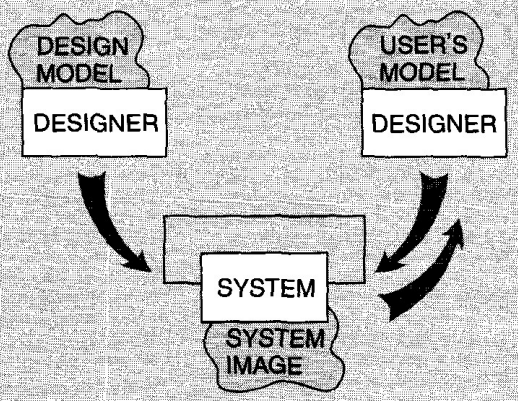


Figure 2.9 Conceptual models.

In an ideal world, all three should map onto each other. Users should be able to carry out their tasks in the way intended by the designer through interacting with the system image, which makes it obvious what to do. However, if the system image does not make the design model clear to the users, it is likely that they will end up with an incorrect understanding of the system, which in turn will make them use the system ineffectively and make errors.

The down side of mixing interaction modes is that the underlying conceptual model can end up being more complex and ambiguous, making it more difficult for the user to understand and learn. For example, some operating and word-processing systems now make it possible for the user to carry out the same activity in a number of different ways (e.g., to delete a file the user can issue a command like CtrlD, speak to the computer by saying "delete file," or drag an icon of the file to the recycle bin). Users will have to learn the different styles to decide which they prefer. Inevitably, the learning curve will be steeper, but in the long run the benefits are that it enables users to decide how they want to interact with the system.

2.4 Interface metaphors

Another way of describing conceptual models is in terms of interface metaphors. By this is meant a conceptual model that has been developed to be similar in some way to ***aspects*** of a physical entity (or entities) but that also has its own behaviors and properties. Such models can be based on an activity or an object or both. As well as being categorized as conceptual models based on objects, the desktop and the spreadsheet are also examples of interface metaphors. Another example of an interface metaphor is a "search engine." The tool has been designed to invite comparison with a physical object—a mechanical engine with several parts working—together with an everyday action—searching by looking through numerous files in many different places to extract relevant information. The functions supported by a search engine also include other features besides those belonging to an engine that searches, such as listing and prioritizing the results of a search. It also does these actions in quite different ways from how a mechanical engine works or how a human being might search a library for books on a given topic. The similarities alluded to by the use of the term "search engine," therefore, are at a very general conceptual level. They are meant to conjure up the essence of the process of finding relevant information, enabling the user to leverage off this "anchor" further understanding of other aspects of the functionality provided.

Interface metaphors are based on conceptual models that combine familiar knowledge with new concepts. As mentioned in Box 2.2, the Star was based on a conceptual model of the familiar knowledge of an office. Paper, folders, filing cabinets, and mailboxes were represented as icons on the screen and were designed to possess some of the properties of their physical counterparts. Dragging a document icon across the desktop screen was seen as equivalent to picking up a piece of paper in the physical world and moving it (but of course is a very different action). Similarly, dragging an electronic document onto an electronic folder was seen as being analogous to placing a physical document into a physical cabinet. In addition, new concepts that were incorporated as part of the desktop metaphor were operations that couldn't be performed in the physical world. For example, electronic files could be placed onto an icon of a printer on the desktop, resulting in the computer printing them out.

BOX 2.4 Why Are Metaphors and Analogies So Popular?

People frequently use analogies and metaphors (here we use the terms interchangeably) as a source of inspiration to understand and explain to others what they are doing or trying to do, in terms that are familiar to them. They are an integral part of human language (Lackoff and Johnson, 1980). They are most commonly used to explain something that is unfamiliar or hard to grasp by way of comparison with something that is familiar and easy to grasp. For example, they are commonly employed in education, where teachers use them to introduce something new to students by comparing the new material to something they already understand. An example is the comparison of human evolution with a game. We are all familiar with the properties of a game: there are rules, each player has a goal to win (or lose), there are heuristics to deal with situations where there are no rules, there is the propensity to cheat when the other players are not looking, and so on. By conjuring up these properties, the analogy helps us begin to understand the more difficult concept of evolution—how it happens, what rules govern it, who cheats, and so.

It is not surprising, therefore, to see how widely metaphors and analogies have been applied in interaction design. Both have been used, in overlapping ways, to conceptualize abstract, hard to imagine, and difficult to articulate computer-based concepts and interactions in more concrete and familiar terms and as graphical visualizations at the interface. This use includes:

- as a way of conceptualizing a particular interaction style, e.g., using the system as a tool
- as a conceptual model that is instantiated as part of an interface, e.g., the desktop metaphor
- as a way of describing computers, eg., the Internet highway
- names for describing specific operations, e.g., "cut" and "paste" commands for deleting and copying objects (analogy taken from the media industry)
- as part of the training material aimed at helping learning, e.g., comparing a word processor with a typewriter.

In many instances, it is hard *not* to use metaphorical terms, as they have become so ingrained in the language we use to express ourselves. This is increasingly the case when talking about computers. Just ask yourself or someone else to describe how the Internet works. Then try doing it without using any metaphors or analogies.

ACTIVITY 2.7 Interface metaphors are often actually composites, i.e., they combine quite different pieces of familiar knowledge with the system functionality. We already mentioned the "search engine" as one such example. Can you think of any others?

Comment Some other examples include:

Scrollbar--combines the concept of a scroll with a bar, as in bar chart

Toolbar--combines the idea of a set of tools with a bar

Portal website—a gateway to a particular collection of pages of networked information

Benefits of interface metaphors

Interface metaphors have proven to be highly successful, providing users with a familiar orienting device and helping them understand and learn how to use a system. People find it easier to learn and talk about what they are doing at the com-

puter interface in terms familiar to them—whether they are computer-phobic or highly experienced programmers. Metaphorically based commands used in Unix, like "lint" and "pipe," have very concrete meanings in everyday language that, when used in the context of the Unix operating system, metaphorically represent some aspect of the operations they refer to. Although their meaning may appear obscure, especially to the novice, they make sense when understood in the context of programming. For example, Unix allows the programmer to send the output of one program to another by using the pipe (|) symbol. Once explained, it is easy to imagine the output from one container going to another via a pipe.

ACTIVITY 2.8 Can you think of any bizarre computing metaphors that have become common parlance whose original source of reference is (or always was) obscure?

Comment A couple of intriguing ones are:

Java—The programing language Java originally was called Oak, but that name had already been taken. It is not clear how the developers moved from Oak to Java. Java is a name commonly associated with coffee. Other Java-based metaphors that have been spawned include Java beans (a reusable software component) and the steaming coffee-cup icon that appears in the top left-hand corner of Java applets.

Bluetooth—Bluetooth is used in a computing context to describe the wireless technology that is able to unite technology, communication, and consumer electronics. The name is taken from King Harald Blue Tooth, who was a 10th century legendary Viking king responsible for uniting Scandinavia and thus getting people to talk to each other.

Opposition to using interface metaphors

A mistake sometimes made by designers is to try to design an interface metaphor to look and behave literally like the physical entity it is being compared with. This misses the point about the benefit of developing interface metaphors. As stressed earlier, they are meant to be used to map familiar to unfamiliar knowledge, enabling users to understand and learn about the new domain. Designing interface metaphors only as literal models of the thing being compared with has understandably led to heavy criticism. One of the most outspoken critics is Ted Nelson (1990) who considers metaphorical interfaces as "using old half-ideas as crutches" (p. 237). Other objections to the use of metaphors in interaction design include:

Breaks the rules. Several commentators have criticized the use of interface metaphors because of the cultural and logical contradictions involved in accommodating the metaphor when instantiated as a GUI. A pet hate is the recycle bin (formerly trash can) that sits on the desktop. Logically and culturally (i.e., in the real world), it should be placed under the desk. If this same rule were followed in the virtual desktop, users would not be able to see the bin because it would be occluded by the desktop surface. A counter-argument to this objection is that it does

not matter whether rules are contravened. Once people understand why the bin is on the desktop, they readily accept that the real-world rule had to be broken. Moreover, the unexpected juxtaposition of the bin on the desktop can draw to the user's attention the additional functionality that it provides.

Too constraining. Another argument against interface metaphors is that they are too constraining, restricting the kinds of computational tasks that would be useful at the interface. An example is trying to open a file that is embedded in several hundreds of files in a directory. Having to scan through hundreds of icons on a desktop or scroll through a list of files seems a very inefficient way of doing this. As discussed earlier, a better way is to allow the user to instruct the computer to open the desired file by typing in its name (assuming they can remember the name of the file).

Conflicts with design principles. By trying to design the interface metaphor to fit in with the constraints of the physical world, designers are forced into making bad design solutions that conflict with basic design principles. Ted Nelson sets up the trash can again as an example of such violation: "a hideous failure of consistency is the garbage can on the Macintosh, which means either "destroy this" or "eject it for safekeeping" (Nelson, 1990).

Not being able to understand the system functionality beyond the metaphor. It has been argued that users may get fixed in their understanding of the system based on the interface metaphor. In so doing, they may find it difficult to see what else can be done with the system beyond the actions suggested by the interface metaphor. Nelson (1990) also argues that the similarity of interface metaphors to any real objects in the world is so tenuous that it gets in the way more than it helps. We would argue the opposite: because the link is tenuous and there are only a certain number of similarities, it enables the user to see both the dissimilarities and how the metaphor has been extended.

Overly literal translation of ***existing bad designs.*** Sometimes designers fall into the trap of trying to create a virtual object to resemble a familiar physical object that is itself badly designed. A well-known example is the virtual calculator, which is designed to look and behave like a physical calculator. The interface of many physical calculators, however, has been poorly designed in the first place, based on poor conceptual models, with excessive use of modes, poor labeling of functions, and difficult-to-manipulate key sequences (Mullet and Sano, 1995). The design of the calculator in Figure 2.10(a) has even gone as far as replicating functions needing shift keys (e.g., deg, oct, and hex), which could have been redesigned as dedicated software buttons. Trying to use a virtual calculator that has been designed to emulate a poorly designed physical calculator is much harder than using the physical device itself. A better approach would have been for the designers to think about how to use the computational power of the computer to support the kinds of tasks people need to do when doing calculations (cf. the spreadsheet design). The calculator in Figure 2.10(b) has tried to do this to some extent, by moving the buttons closer to each other (minimizing the amount of mousing) and providing flexible display modes with one-to-one mappings with different functions.

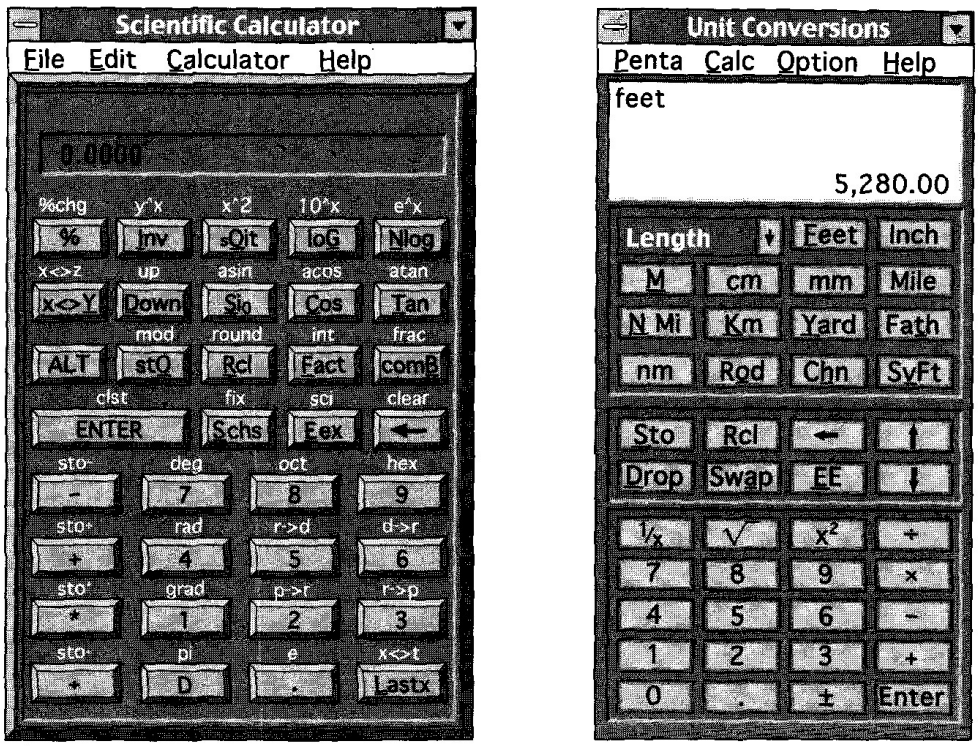


Figure 2.10 Two virtual calculators where (a) has been designed too literally and (b) more appropriately for a computer screen.

Limits the designer's imagination in conjuring up new paradigms and models. Designers may fixate on "tired" ideas, based on well known technologies, that they know people are very familiar with. Examples include travel and books for representing interaction with the web and hypermedia. One of the dangers of always looking backwards is that it restricts the designer in thinking of what new functionality to provide. For example, Gentner and Nielsen (1996) discuss how they used a book metaphor for designing the user interface to Sun Microsystems' online documentation. In hindsight they realized how it had blinkered them in organizing the online material, preventing them from introducing desirable functions such as the ability to reorder chapters according to their relevance scores after being searched.

Clearly, there are pitfalls in using interface metaphors in interaction design. Indeed, this approach has led to some badly designed conceptual models, that have resulted in confusion and frustration. However, this does not have to be the case. Provided designers are aware of the dangers and try to develop interface metaphors that effectively combine familiar knowledge with new functionality in a meaningful way, then many of the above problems can be avoided. Moreover, as we have seen with the spreadsheet example, the use of analogy as a basis for a conceptual model can be very innovative and successful, opening up the realm of computers and their applications to a greater diversity of people.

ACTIVITY 2.9 Examine a web browser interface and describe the various forms of analogy and composite interface metaphors that have been used in its design. What familiar knowledge has been combined with new functionality?

Comment Many aspects of a web browser have been combined to create a composite interface metaphor:

- a range of toolbars, such as a button bar, navigation bar, favorite bar, history bar
- tabs, menus, organizers
- search engines, guides
- bookmarks, favorites
- icons for familiar objects like stop lights, home

These have been combined with other operations and functions, including saving, searching, downloading, listing, and navigating.

2.5 Interaction paradigms

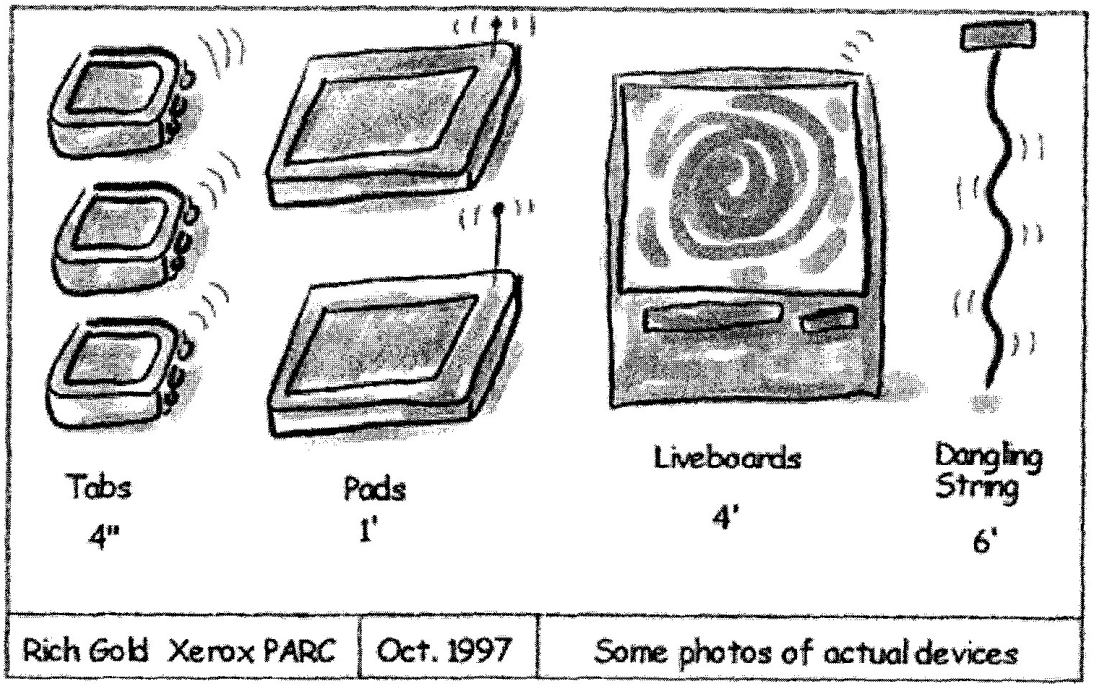
At a more general level, another source of inspiration for informing the design of a conceptual model is an interaction paradigm. By this it is meant a particular philosophy or way of thinking about interaction design. It is intended to orient designers to the kinds of questions they need to ask. For many years the prevailing paradigm in interaction design was to develop applications for the desktop—intended to be used by single users sitting in front of a CPU, monitor, keyboard and mouse. A dominant part of this approach was to design software applications that would run using a GUI or WIMP interface (windows, icons, mouse and pull-down menus, alternatively referred to as windows, icons, menus and pointers).

As mentioned earlier, a recent trend has been to promote paradigms that move "beyond the desktop." With the advent of wireless, mobile, and handheld technologies, developers started designing applications that could be used in a diversity of ways besides running only on an individual's desktop machine. For example, in September, 2000, the clothes company Levis, with the Dutch electronics company Philips, started selling the first commercial e-jacket—incorporating wires into the lining of the jacket to create a body-area network (BAN) for hooking up various devices, e.g., mobile phone, MP3, microphone, and headphone (see Figure 1.2(iii) in Color Plate 1). If the phone rings, the MP3 player cuts out the music automatically to let the wearer listen to the call. Another innovation was handheld interactive devices, like the PalmPilot, for which a range of applications were programmed. One was to program the PalmPilot as a multipurpose identity key, allowing guests to check in to certain hotels and enter their room without having to interact with the receptionist at the front desk.

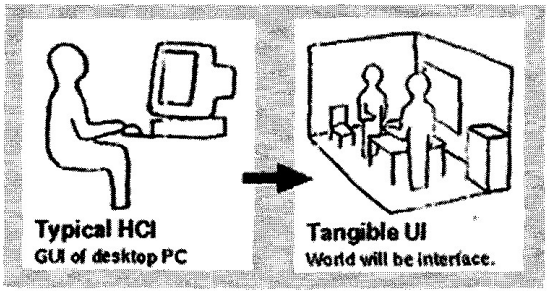
A number of alternative interaction paradigms have been proposed by researchers intended to guide future interaction design and system development (see Figure 2.11). These include:

- ubiquitous computing (technology embedded in the environment)
- pervasive computing (seamless integration of technologies)
- wearable computing (or wearables)

(a)



(b)



(c)

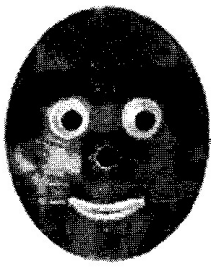


Figure 2.11 Examples of new interaction paradigms: (a) Some of the original devices developed as part of the ubiquitous computing paradigm. Tabs are small hand-sized wireless computers which know where they are and who they are with. Pads are paper-sized devices connected to the system via radio. They know where they are and who they are with. Liveboards are large wall sized devices. The "Dangling String" created by artist Natalie Jeremijenko was attached directly to the ethernet that ran overhead in the ceiling. It spun around depending on the level of digital traffic.

(b) Ishii and Ulmer, MIT Lab (1997) Tangible bits: from GUIs of desktop PCs to Tangible User Interfaces. The paradigm is concerned with establishing a new type of HCI called "Tangible User Interfaces" (TUIs). TUIs augment the real physical world by coupling digital information to everyday physical objects and environments.

(c) Affective Computing: The project, called "BlueEyes," is creating devices with embedded technology that gather information about people. This face (with movable eyebrows, eyes and mouth) tracks your movements and facial expressions and responds accordingly.

- tangible bits, augmented reality, and physical/virtual integration
- attentive environments (computers attend to user's needs)
- the Workaday World (social aspects of technology use)

Ubiquitous computing ("ubicomp"). The late Mark Weiser (1991), an influential visionary, proposed the interaction paradigm of ubiquitous computing (Figure 2.11). His vision was for computers to disappear into the environment so that we would be no longer aware of them and would use them without thinking about them. As part of this process, they should "invisibly" enhance the world that already exists rather than create artificial ones. Existing computing technology, e.g., multimedia-based systems and virtual reality, currently do not allow us to do this. Instead, we are forced to focus our attention on the multimedia representations on the screen (e.g., buttons, menus, scrollbars) or to move around in a virtual simulated world, manipulating virtual objects.

So, how can technologies be designed to disappear into the background? Weiser did not mean ubiquity in the sense of simply making computers portable so that they can be moved from the desk into our pockets or used on trains or in bed. He meant that technology be designed to be integrated seamlessly into the physical world in ways that ***extend*** human capabilities. One of his prototypes was a "tabs, pads, and boards" setup whereby hundreds of computer devices equivalent in size to post-it notes, sheets of paper, and blackboards would be embedded in offices. Like the spreadsheet, such devices are assumed to be easy to use, because they capitalize on existing knowledge about how to interact and use everyday objects. Also like the spreadsheet, they provide much greater computational power. One of Weiser's ideas was that the tabs be connected to one another, enabling them to become multipurpose, including acting as a calendar, diary, identification card, and an interactive device to be used with a PC.

> ***Ubiquitous computing will produce nothing fundamentally new, but by making everything faster and easier to do, with less strain and fewer mental gymnastics, it will transform what is apparently possible (Weiser, 1991, p. 940).***

Pervasive computing. Pervasive computing is a direct follow-on of ideas arising from ubiquitous computing. The idea is that people should be able to access and interact with information any place and any time, using a seamless integration of technologies. Such technologies are often referred to as smart devices or information appliances—designed to perform a particular activity. Commercial products include cell phones and handheld devices, like PalmPilots. On the domestic front, other examples currently being prototyped include intelligent fridges that signal the user when stocks are low, interactive microwave ovens that allow users to access information from the web while cooking, and smart pans that beep when the food is cooked.

Wearable computing. Many of the ideas behind ubiquitous computing have since inspired other researchers to develop technologies that are part of the environment. The MIT Media Lab has created several such innovations. One example is wearable computing (Mann, 1996). The combination of multimedia and wireless

communication presented many opportunities for thinking about how to embed such technologies on people in the clothes they wear. Jewelry, head-mounted caps, glasses, shoes, and jackets have all been experimented with to provide the user with a means of interacting with digital information while on the move in the physical world. Applications that have been developed include automatic diaries that keep users up to date on what is happening and what they need to do throughout the day, and tour guides that inform users of relevant information as they walk through an exhibition and other public places (Rhodes et al., 1999).

Tangible bits, augmented reality, and physical/virtual integration. Another development that has evolved from ubiquitous computing is tangible user interfaces or tangible bits (Ishii and Ullmer, 1997). The focus of this paradigm is the "integration of computational augmentations into the physical environment", in other words, finding ways to combine digital information with physical objects and surfaces (e.g., buildings) to allow people to carry out their everyday activities. Examples include physical books embedded with digital information, greeting cards that play a digital animation when opened, and physical bricks attached to virtual objects that when grasped have a similar effect on the virtual objects. Another illustration of this approach is the one described in Chapter 1 of an *enjoyable* interface, in which a person could use a physical hammer to hit a physical key with corresponding virtual representations of the action being displayed on a screen.

Another part of this paradigm is augmented reality, where virtual representations are superimposed on physical devices and objects (as shown in Figure 2.1 on Color Plate 2). Bridging the gulf between physical and virtual worlds is also currently undergoing much research. One of the earlier precursors of this work was the Digital Desk (Wellner, 1993). Physical office tools, like books, documents and paper, were integrated with virtual representations, using projectors and video cameras. Both virtual and real documents were seamlessly combined.

Attentive environments and transparent computing. This interaction paradigm proposes that the computer attend to user's needs through anticipating what the user wants to do. Instead of users being in control, deciding what they want to do and where to go, the burden should be shifted onto the computer. In this sense the mode of interaction is much more implicit: computer interfaces respond to the user's expressions and gestures. Sensor-rich environments are used to detect the user's current state and needs. For example, cameras can detect where people are looking on a screen and decide what to display accordingly. The system should be able to determine when someone wants to make a call and which websites they want to visit at particular times. IBM's BlueEyes project is developing a range of computational devices that use non-obtrusive sensing technology, including videos and microphones, to track and identify users' actions. This information is then analyzed with respect to where users are looking, what they are doing, their gestures, and their facial expressions. In turn, this is coded in terms of the users' physical, emotional or informational state and is then used to determine what information they would like. For example, a BlueEyes-enabled computer could become active when a user first walks into a room, firing up any new email messages that have arrived. If the user shakes his or her head, it would be interpreted by the computer as "I don't want to read them," and instead show a listing of their appointments for that day.

The Workaday World. In the new paradigms mentioned above, the emphasis is on exploring how technological devices can be linked with each other and digital information in novel ways that allow people to do things they could not do before. In contrast, the Workaday World paradigm is driven primarily by conceptual and mundane concerns. It was proposed by Tom Moran and Bob Anderson (1990), when working at Xerox PARC. They were particularly concerned with the need to understand the social aspects of technology use in a way that could be useful for designers. The Workaday World paradigm focuses on the essential character of the workplace in terms of people's everyday activities, relationships, knowledge, and resources. It seeks to unravel the "set of patterns that convey the richness of the settings in which technologies live—the complex, unpredictable, multiform relationships that hold among the various aspects of working life" (p. 384).

2.6 From conceptual models to physical design

As we emphasize throughout this book, interaction design is an iterative process. It involves cycling through various design processes at different levels of detail. Primarily it involves: thinking through a design problem, understanding the user's needs, coming up with possible conceptual models, prototyping them, evaluating them with respect to usability and user experience goals, thinking about the design implications of the evaluation studies, making changes to the prototypes with respect to these, evaluating the changed prototypes, thinking through whether the changes have improved the interface and interaction, and so on. Interaction design may also require going back to the original data to gather and check the requirements. Throughout the iterations, it is important to think through and understand whether the conceptual model being developed is working in the way intended and to ensure that it is supporting the user's tasks.

Throughout this book we describe the way you should go about ***doing*** interaction design. Each iteration should involve ***progressing through*** the design in more depth. A first pass through an iteration should involve essentially thinking about the problem space and identifying some initial user requirements. A second pass should involve more extensive information gathering about users' needs and the problems they experience with the way they currently carry out their activities (see Chapter 7). A third pass should continue explicating the requirements, leading to thinking through possible conceptual models that would be appropriate (see Chapter 8). A fourth pass should begin "fleshing out" some of these using a variety of user-centered methods. A number of user-centered methods can be used to create prototypes of the potential candidates. These include using storyboarding to show how the interaction between the users and the system will take place and the laying out of cards and post-it notes to show the possible structure of and navigation through a website. Throughout the process, the various prototypes of the conceptual models should be evaluated to see if they meet users' needs. Informally asking users what they think is always a good starting point (see Chapter 12). A number of other techniques can also be used at different stages of the development of the prototypes, depending on the particular information required (see Chapters 13 and 14).

Many issues will need to be addressed when developing and testing initial prototypes of conceptual models. These include:

- the way information is to be presented and interacted with at the interface
- what combinations of media to use (e.g., whether to use sound and animations)
- the kind of feedback that will be provided
- what combinations of input and output devices to use (e.g., whether to use speech, keyboard plus mouse, handwriting recognition)
- whether to provide agents and in what format
- whether to design operations to be hardwired and activated through physical buttons or to represent them on the screen as part of the software
- what kinds of help to provide and in what format

While working through these design decisions about the nature of the interaction to be supported, issues concerning the actual physical design will need to be addressed. These will often fall out of the conceptual decisions about the way information is to be represented, the kind of media to be used, and so on. For example, these would typically include:

- ***information presentation***
  - –which dialogs and interaction styles to use (e.g., form fill-ins, speech input, menus)
  - –how to structure items in graphical objects, like windows, dialog boxes and menus (e.g., how many items, where to place them in relation to each other)
- ***feedback***
  - –what navigation mechanisms to provide (e.g., forward and backward buttons)
- ***media combination***
  - –which kinds of icons to use

Many of these physical design decisions will be specific to the interactive product being built. For example, designing a calendar application intended to be used by business people to run on a handheld computer will have quite different constraints and concerns from designing a tool for scheduling trains to run over a large network, intended to be used by a team of operators via multiple large displays. The way the information will be structured, the kinds of graphical representations that will be appropriate, and the layout of the graphics on the screens will be quite different.

These kinds of design decisions are very practical, needing user testing to ensure that they meet with the usability goals. It is likely that numerous trade-offs will surface, so it is important to recognize that there is no right or wrong way to resolve these. Each decision has to be weighed with respect to the others. For example, if you decide that a good way of providing visibility for the calendar application on the handheld device is to have a set of "soft" navigation buttons permanently as

DILEMMA Realism versus Abstraction?

One of the challenges facing interaction designers is whether to use realism or abstraction when designing an interface to instantiate their conceptual model. By this it is meant designing objects either (i) to give the illusion of behaving and looking like real-world counterparts or (ii) to appear as simply abstractions of the objects being represented. This concern is particularly relevant when fleshing out conceptual models that are deliberately based on an analogy to some aspect of the real world. For example, is it preferable to design a desktop to look like a real desktop, a virtual house to look like a real house, or a virtual living room to look like a real living room? Or, alternatively, is it more effective to design representations of the conceptual model as simple abstract renditions, depicting only a few salient features?

We already discussed in Chapter 1 the problems of trying to design graphical interfaces with affordances. Here, we consider more generally the dilemma of using realism at the interface. One of the main benefits of using realism is that it can enable people, especially computer phobics and novices, to feel *more comfortable* when first learning an application. The reason for this is that such representations can readily tap into people's understanding of the physical world. Hence, realistic interfaces can help users initially understand the underlying conceptual model. In contrast, overly schematic and abstract representations can appear to be too computer-like and off-putting to the newcomer. The advantage of these kinds of more abstract interfaces is that they are often more efficient to use. Furthermore, the more experienced users become, the more they may find comfortable interfaces no longer to their liking. A dilemma facing designers, therefore, is deciding between designing interfaces to make novice users feel comfortable (but more experienced users less comfortable) versus designing interfaces to be effective for more experienced users (but maybe harder to learn by novices).

One of the earliest attempts at using realism at the interface was General Magic's office system Magic Cap, which was rendered in 3D. To achieve this degree of realism required using various perceptual cues such as perspective, shadowing, and shading. The result of their efforts was a rather cute interface (see Figure 2.12). Although their intentions were well-grounded, the outcome was less successful. Many people commented on how childish and gawky it looked, having the appearance of illustrations in a children's picture book rather than a work-based application.

Mullet and Sano (1995) also point out how a 3D rendition of an object like a desk nearly always suffers from both an unnatural point of view and an awkward rendering style that ironically destroy the impression of being in a real physical space. One reason for this is that 3D depictions conflict with the effective use of display space, especially when 2D editing tasks need to be performed. As can be seen in Figure 2.12, these kinds of tasks have been represented as "flat" buttons that appear to be floating in front of the desk (e.g., mail, program manager, task manager).

In certain kinds of applications, using realism can be very effective for both novices and experienced users. Video games fall into this category, especially those where users have to react rapidly to dynamic events that happen in a virtual world in real time, say flying a plane or playing a game of virtual football. Making the characters in the game resemble humans in the way they look, move, dress, and gesture also makes them seem more convincing and lifelike, enhancing the enjoyment and fun factor (see Figure 2.13).

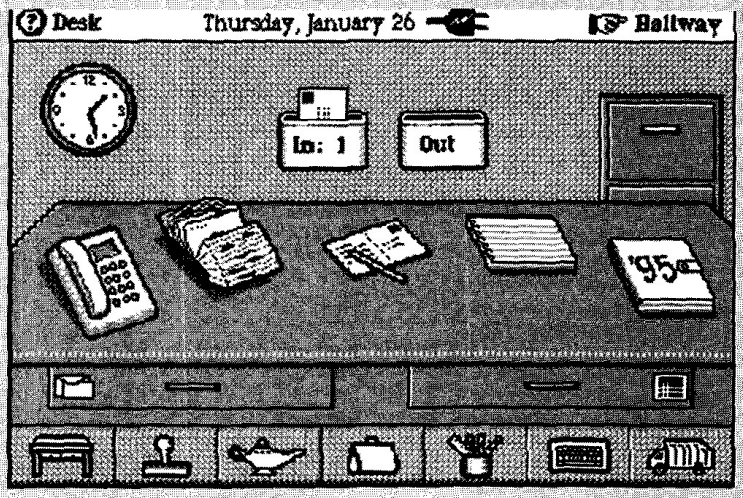


Figure 2.12 Magic Cap's 3D desktop interface.

Figure 2.13 3D avatars in computer games: A screenshot from The Sims World.

part of the visual display, you then need to consider the consequences of doing this for the rest of the information that needs to be interacted with. Will it still be possible to structure the display to show the calendar as days in a week or a month, all on one screen?

This part of the design process is highly dependent on the context and essentially involves lots of juggling between design decisions. If you visit our website you can try out some of the interactivities provided, where you have to make such decisions when designing the physical layout for various interfaces. Here, we provide the background and rationale that can help you make appropriate choices when faced with a series of design decisions (primarily Chapters 3–5 and 8). For example, we explain why you shouldn't cram a screen full of information; why certain techniques are better than others for helping users remember how to carry out their tasks at the interface; and why certain kinds of agents appear more believable than others.

Assignment

The aim of this assignment is for you to think about the appropriateness of different kinds of conceptual model that have been designed for similar kinds of physical and electronic artifacts.

(a) Describe the conceptual model that underlie the design of:

- a personal pocket-sized calendar/diary (one week to a page)
- a wall calendar (one month to a page, usually with a picture/photo)
- a wall planner (displaying the whole year)

What is the main kind of activity and object they are based on? How do they differ for each of the three artifacts? What metaphors have been used in the design of their physical interface (think about the way time is conceptualized for each of them)? Do users understand the conceptual models these are based on in the ways intended (ask a few people to explain how they use them)? Do they match the different user needs?

(b) Now describe the conceptual models that underlie the design of:

- an electronic ***personal*** calendar found on a personal organizer or handheld computer
- a ***shared*** calendar found on the web

How do they differ from the equivalent physical artifacts? What new functionality has been provided? What interface metaphors have been used? Are the functions and interface metaphor well integrated? What problems do users have with these interactive kinds of calendars? Why do you think this is?

Summary

This chapter has explained the importance of conceptualizing interaction design before trying to build anything. It has stressed throughout the need always to be clear and explicit about the rationale and assumptions behind any design decision made. It described a taxonomy of conceptual models and the different properties of each. It also discussed interface metaphors and interaction paradigms as other ways of informing the design of conceptual models.

Key points

- It is important to have a good understanding of the problem space, specifying what it is you are doing, why and how it will support users in the way intended.
- A fundamental aspect of interaction design is to develop a conceptual model.
- There are various kinds of conceptual models that are categorized according to the activity or object they are based on.
- Interaction modes (e.g., conversing, instructing) provide a structure for thinking about which conceptual model to develop.
- Interaction styles (e.g., menus, form fill-ins) are specific kinds of interfaces that should be decided upon after the conceptual model has been chosen.
- Decisions about conceptual design also should be made before commencing any physical design (e.g., designing an icon).
- Interface metaphors are commonly used as part of a conceptual model.
- Many interactive systems are based on a hybrid conceptual model. Such models can provide more flexibility, but this can make them harder to learn.
- 3D realism is not necessarily better than 2D or other forms of representation when instantiating a conceptual model: what is most effective depends on the users' activities when interacting with a system.
- General interaction paradigms, like WIMP and ubiquitous computing, provide a particular way of thinking about how to design a conceptual model.

Further reading

LAUREL, B. (1990) (ed.) *The Art of Human Computer Design* has a number of papers on conceptual models and interface metaphors. Two that are definitely worth reading are: Tom Erickson, "Working with interface metaphors" (pp. 65–74), which is a practical hands-on guide to designing interface metaphors (covered later in this book), and Ted Nelson's polemic, "The right way to think about software design" (pp. 229–234), which is a scathing attack on the use of interface metaphors.

JOHNSON, M. AND LAKOFF, G. (1980) ***Metaphors We Live By.*** The University of Chicago Press. Those wanting to find out more about how metaphors are used in everyday conversations should take a look at this text.

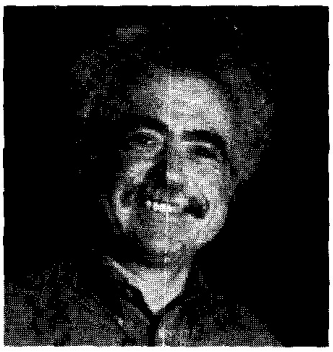
There are many good articles on the topic of interface agents. A classic is:

LANIER, J. (1995) Agents of alienation, ***ACM Interactions,*** 2(3), 66–72. *The Art of Human Computer Design* also provides several thought-provoking articles, including one called "Interface agents: metaphors with character" by Brenda Laurel (pp. 355–366) and another called "Guides: characterizing the interface" by Tim Oren et al. (pp. 367–382).

BANNON, L. (1977) "Problems in human-machine interaction and communication." ***Proc HCI'97,*** San Francisco. Bannon presents a critical review of the agent approach to interface design.

MIT's Media Lab (www.media.mit.edu) is a good starting place to find out what is currently happening in the world of agents, wearables, and other new interaction paradigms.

INTERVIEW with Terry Winograd



Terry Winograd is a professor of computer science at Stanford University. He has done extensive research and writing on the design of human-computer interaction. His early research on natural language understanding by computers was a milestone in artificial intelligence, and he has written two books and numerous articles on hat topic. His book, Bringing Design *to Software*, brings together the perspectives of a number of leading researchers and designers. See Color Plate 2 for an example of his latest research.

YR: Tell me about your background and how you moved into interaction design.

TW: I got into interaction design through a couple of intermediate steps. I started out doing research into artificial intelligence. I became interested in how people interact with computers, in particular, when using ordinary language. It became clear after years of working on that, however, that the computer was a long way off from matching human abilities. Moreover, using natural language with a computer when it doesn't really understand you can be very frustrating and in fact a very bad way to interact with it. So, rather than trying to get the computer to imitate the person, I became interested in other ways of taking advantage of what the computer can do well and what the person can do well. That led me into the general field of HCI. As I began to look at what was going on in that field and to study it, it became clear that it was not the same as other areas of computer science. The key issues were about how the technology fits with what people could do and what they wanted to do. In contrast, most of computer science is really dominated by how the mechanisms operate.

I was very attracted to thinking more in the style of design disciplines, like product design, urban design, architecture, and so on. I realized that there was an approach that you might call a design way, that puts the technical asspects into the background with respect to understanding the interaction. Through looking at these design disciplines, I realized that there was something unique about interaction design, which is that it has a dialogic temporal element. By this I mean a human dialog not in the sense of using ordinary language, but in the sense of thinking about the sequence and the flow of interaction. So I think interaction design is about designing a space for people, where that space has to have a temporal flow. It has to have a dialog with the person.

YR: Could you tell me a bit more about what you think is involved in interaction design?

TW: One of the biggest influences is product design. I think that interaction design overlaps with it, because they both take a very strong user-oriented view. Both are concerned with finding a user group, understanding their needs, then using that understanding to come up with new ideas. They may be ones that the users don't even realize they need. It is then a matter of trying to translate who it is, what they are doing, and why they are doing it into possible innovations. In the case of product design it is products. In the case of interaction design it is the way that the computer system interacts with the person.

YR. What do you think are important inputs into the design process?

TW: One of the characteristics of design fields as opposed to traditional engineering fields is that there is much more dependence on case studies and examples than on formulas. Whereas an engineer knows how to calculate something, an architect or a designer is working in a tradition where there is a history over time of other things people have done. People have said that the secret of great design is to know what to steal and to know when some element or some way of doing things that worked before will be appropriate to your setting and then adapt it. Of course you can't apply it directly, so I think a big part of doing good design is experience and exposure. You have to have seen a lot of things in practice and understood what is good and bad about them, to then use these to inform your design.

YR: How do you see the relationship between studying interaction design and the practice of it? Is there a good dialog between research and practice?

TW: Academic study of interaction design is a tricky area because so much of it depends on a kind of tacit knowledge that comes through experience and

exposure. It is not the kind of thing you can set down easily as, say, you can scientific formulas. A lot of design tends to be methodological. It is not about the design per se but is more about how you go about doing design, in particular, knowing what are the appropriate steps to take and how you put them together.

YR: How do you see the field of interaction design taking on board the current explosion in new technologies—for example mobile, ubiquitous, infrared, and so on? Is it different, say, from 20 years ago when it was just about designing software applications to sit on the desktop?

TW: I think a real change in people's thinking has been to move from interface design to interaction design. This has been pushed by the fact that we do have all kinds of devices nowadays. Interface design used to mean graphical interfaces, which meant designing menus and other widgets. But now when you're talking about handheld devices, gesture interfaces, telephone interfaces and so on, it is clear that you can't focus just on the widgets. The widgets may be part of any one of these devices but the design thinking as a whole has to focus on the interaction.

YR: What advice would you give to a student coming into the field on what they should be learning and looking for?

TW: I think a student who wants to learn this field should think of it as a kind of dual process, that is what Donald Schon calls "reflection in action," needing both the action and the reflection. It is important to have experience with trying to build things. That experience can be from outside work, projects, and courses where you are actually engaged in making something work. At the same time you need to be able to step back and look at it not as "What do I need to do next?" but from the perspective of what you are doing and how that fits into the larger picture.

YR: Are there any classic case studies that stand out as good exemplars of interaction design?

TW: You need to understand what has been important in the past. I still use the Xerox Star as an exemplar because so much of what we use today was there. When you go back to look at the Star you see it in the context of when it was first created. I also think some exemplars that are very interesting are ones that never actually succeeded commercially. For example, I use the PenPoint system that was developed for pen computers by Go. Again, they were thinking fresh. They set out to do something different and they were much more conscious of the design issues than somebody who was simply adapting the next version of something that already existed. PalmPilot is another good example, because they looked at the problem in a different way to make something work. Another interesting exemplar, which other people may not agree with, is Microsoft Bob--not because it was a successful program, because it wasn't, but because it was a first exploration of a certain style of interaction, using animated agents. You can see very clearly from these exemplars what design trade-offs the designers were making and why and then you can look at the consequences.

YR: Finally, what are the biggest challenges facing people working in this area?

TW: I think one of the biggest challenges is what Pelle Ehn calls the dialectic between tradition and transcendence. That is, people work and live in certain ways already, and they understand how to adapt that within a small range, but they don't have an understanding or a feel for what it would mean to make a radical change, for example, to change their way of doing business on the Internet before it was around, or to change their way of writing from pen and paper when word processors weren't around. I think what the designer is trying to do is envision things for users that the users can't yet envision. The hard part is not fixing little problems, but designing things that are both innovative and that work.