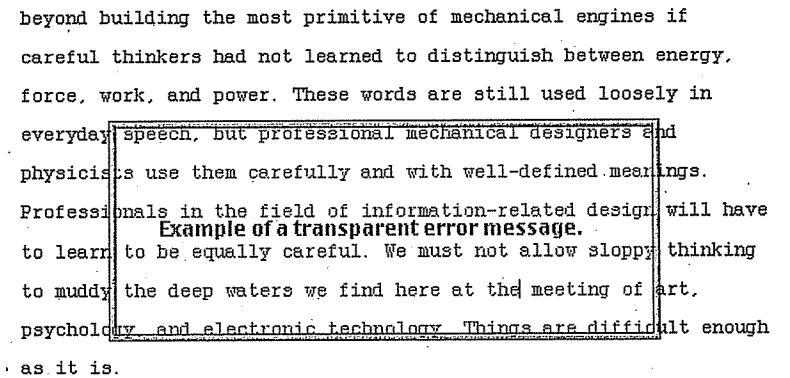


Because it would be a waste of time and display space to designate a special location for typing commands, a command should be typed wherever the cursor happens to be at the moment the need for the command becomes apparent. The typed command name should be deleted after being executed, so that command names are not left scattered among your content. On the other hand, when you execute a command from a list of commands, you do not want the command to disappear; in effect, the list is a menu. To create such a menu, nothing more is needed than to type a list of commands, select the list, and then use a command, perhaps called Make Menu, that can change them to a distinctive style typically used only for commands and, at the same time, lock the list of items so that it cannot be inadvertently changed.

Here are some other commands that change the state of text. One convenience is to be able to simply lock text or other content, with a **Lock** command. Locked content can be seen, selected, and copied but cannot be changed or moved. The inverse, **Unlock**, can be applied to selected content and will unlock the selection if the selection was locked. (Otherwise, it has no effect; it must not toggle.) Another command, **Lock with Password**, locks the old selection, using the current selection as the password. It, too, has an inverse (**Unlock with Password**). Locked content has a wide range of utility; for example, it can be used to create forms to be filled out. The fixed parts of the form are locked or password locked. Simple locking prevents accidental changes; password locking prevents unauthorized changes. If the online instruction manual to a computer system were included as part of the text that the computer initially came with—which is not a bad idea—the online manual would probably be password locked at the factory.

Screen Lock and **Screen Unlock** lock and unlock the position of selected objects with respect to the display. With this facility, you can create menus that stay in place on the display as other objects move underneath them: You could simulate today's fixed-position menus. (Whether this is a good use of this facility is a different question.) To use this command, you would place the object as desired with respect to the screen, select it, and use the Screen Lock command. There could be a password-protected version of this as well, for occasions when it is undesirable for the user to redesign the menus.

Another useful command is one that controls the transparency of a selection. In some situations, such as when displaying error messages, it is useful to render the selection transparent enough so that underlying material can be seen and operated through (Figure 5.2). Similarly, another command



beyond building the most primitive of mechanical engines if careful thinkers had not learned to distinguish between energy, force, work, and power. These words are still used loosely in everyday speech, but professional mechanical designers and physicists use them carefully and with well-defined meanings. Professionals in the field of information-related design will have to learn to be equally careful. We must not allow sloppy thinking to muddy the deep waters we find here at the meeting of art, psychology and electronic technology. Things are difficult enough as it is.

Figure 5.2. A transparent error message over background text has an efficiency of 1.

could specify whether an object hides or is hidden by or is viewed through another object. A transparent message box can fade gradually instead of abruptly, giving you time to notice it. A document that stores all messages for later review is essential.

Because you can click through a transparent dialog box, removing it requires no keystrokes; it is modeless and highly efficient, with an efficiency of 1. As with any method, this idea has its limitations and can be overused; a deluge of unnecessary messages is still distracting, even if they disappear when the user proceeds. In accord with the principle of visibility, a visible distinction should be provided to allow a user to determine whether text is locked, screen locked, locked with a password, and so forth.

A humane-interface principle is that the system itself should be built out of the same kind of pieces with which you are familiar from your everyday use of the system. This makes your product less forbidding.

5-3 File Names and Structures

Mankind are more disposed to suffer, while evils are sufferable, than to right themselves by abolishing the forms to which they are accustomed.

—Thomas Jefferson, Declaration of Independence of the United States of America

Because the restriction of file name length to 8 characters in earlier systems was truly dreadful, the 31-character maximum length offered by the Macintosh looked like heaven to many users. But this modification was just a different-size straitjacket. Aside from the real limits of the hardware, an interface should have few, if any, fixed-length limits. An interface should use

The Humane Interface, Rasmus, Addison Wesley, 2000

dynamic memory allocation, linked lists, hashing, or whatever techniques seem best but should never present a user with software implementation limitations, such as “you can have up to 255 categories” or “paragraphs can be up to 32,000 characters.”²

What is a file name? From the user’s perspective, it is a handle with which to grab a file. As we know from long experience, file names do not work as one would expect; they are an impediment when you want to save, futile when you want to find. Let me be more specific: File names are bothersome when you are about to save work, because you have to stop in the middle of your activity, which is trying to store your work away, and invent a file name.³ Creating names is an onerous task: You are required to invent, on the spot and in a few moments, a name that is unique, memorable, and within the naming conventions of the system you are using. Furthermore, at that moment, the question of a file name is not your locus of attention; preserving your work is. File names are also a nuisance when you have to retrieve a file. The name you thought up was probably not particularly memorable, and you probably forgot it after a few weeks (or less). I, for one, can rarely remember a file name unless it is quite recent; even looking through lists of file names is frustrating. Just what is in that file that is labeled “notes ybn 32”? The name seemed so clever and memorable when I created it. Then, too, many files are nearly the same. How many different, creative, readily remembered names can you think up for letters to your accountant about last year’s taxes? Filing them by date may be useful, but how many of us remember that the letter about the deduction for the company truck was written on August 14?

Having to name files increases the mental burden on the user. Giving a file a name does nothing more than add a few characters to the file, and you are required to remember the file by that one tiny portion and by nothing else. I count this as one of the major horrors visited upon us by conventional computer systems. Many information appliances have also adopted this inhumane methodology.

There should be no distinction between a file name and a file. A human mind can more effectively use a fast, whole-text search engine, so

2. A word processor I once used quite a while ago had this paragraph size limit. I exceeded it as soon as I inserted a photograph into a paragraph; when I spoke to the designers, they admitted they had never thought of that. The moral: Never put in a fixed limit because it makes the software easier to write; it will always be too small.

3. This is in the context of present systems. You should never have to explicitly perform a save in future, humane systems.

that any word or phrase from the file can serve as a key to it. (Eventually, we’d want more: A request for “a letter about dragonflies” would cause a search that looked for something that had the form of a letter and looked not only for the word *dragonfly* but also for related terms or expressions, such as *Odonata*—in case dragonflies had been referred to by their scientific name—and if no instances of such letters were found, the search would look for nonletter documents with that content, and so forth, extending out to networked computers and to the Internet.) You do not remember the content of “Letter 12/21/92 to Jim” when you see that title, but you do remember that you once wrote to Jim about the blue Edsel that ran across your eyeglasses. A search on *Edsel* is likely to find only one or two entries on your whole system—unless you are an Edsel fancier, in which case you would probably choose another pattern on which to search. An unlimited-length file name is a file. *The content of a text file is its own best name.*

Graphics and sound files often require names; Section 6-2 discusses an approach that avoids the memory burden that traditional file structures impose for nontext files. Aside from nontext files, given a fast whole-text search, file names—one whole species of unnecessary entities—can be eliminated. With the removal of file names go all the mechanisms for dealing with file names, such as directories of file names, rules for editing file names, and syntactic restrictions on file names. When file names are eliminated, a significant mental burden and much internal machinery—machinery that is currently part of what you have to learn and what programmers have to implement—vanishes.

The best interface to a whole-text search is interactive, where you see each found instance in context as it is found. In this interface, as soon as you see what you want, you are there. Some systems present copies of the found instances in the line that contains them (Drori 1998). However, this method is not as efficient as the first kind of search, because you must then perform a second operation to reach the instance itself; for example, you must click on the copy of the desired instance.

For users who insist on a system that looks like conventional file structures, there can be a command that creates an “information document” or an extra page at the end of each document when the document is selected and the command applied. The information document or page would contain such information as the date and time the document was created or modified, a revision history, the length of the first document, or whatever information seems of value. The software to implement such a command would have to acquire and store the necessary information invisibly to the user. Various vendors might provide different facilities, depending

on user needs. For users who wish to hold on to their old ways, a vendor could even go so far as to create utilities that create documents that look and act just like the annoying directories we now have.

Another source of organization, one that is easier to learn and to use than traditional file systems, comes from the inherent hierarchical structure of many natural languages: Words are separated by spaces. Sentences, or sequences of words, are separated by one of a small number of delimiters, followed by a space. (In English, these delimiters include periods, question marks, and exclamation points.) Paragraphs, or sequences of sentences, are separated by at least one explicit Return. Explicit page characters, or page breaks, separate chapters or whatever you wish to call the next level of organization.

In a consistent system, page breaks should be characters and, unlike most present systems, they should behave—in terms of insertion, deletion, and searching—just as any other characters do. As with Return, there may be implicit page breaks to accommodate the fixed length of physical pages, but these are not part of the content.⁴

There is good reason not to stop here in the hierarchy as many present systems do. Documents are sequences of pages separated by document characters, each typable, searchable, and deletable, just as is any other character. There can be higher delimiters, such as folder and volume characters, even section and library delimiters, but the number of levels depends on the size of the data. A set of two consecutive document characters makes a fine delimiter for sets of documents. If more levels of organization are needed, three or four consecutive document characters could be used as delimiters. It can be easier to repeatedly tap the Document key, even four times, than to type such rarely used keys as Folder, Volume, and Library. This convention also prevents an explosion of new keys on the keyboard. It is important for all delimiter characters to have dedicated keys, for otherwise, they would not behave as do all other typable characters. That is, we should not use a key-stroke for, say, Return, and then use an Insert Page Break command; we must have a page character.

Because the various separator characters would behave exactly as did all other characters, there would be no need to teach users how to do searches for them. An individual who insists on having explicit document names can adopt the personal convention of placing the desired names

4. Dr. James Winter, at Information Appliance, further unified the structure by pointing out that English already has the same kind of hierarchical, character-delimited structure proposed for the higher levels of organization.

immediately following document characters. To find a document with the name “Dogs of Asia,” you would search for a string that began with a document character followed by *Dogs of Asia*. Such a search would ignore all instances of *Dogs of Asia* except those used as document names. If you wanted a directory, there could be a command that would assemble a document comprising all instances of strings consisting of a document character, followed by any other characters, up to and including the first Return or higher delimiter.

Eliminating hierarchical file structures does not mean that you have to give up structuring your stored information. Nothing prevents you from creating tables of contents and indexes or putting all of your letters to Uncle Albert and Aunt Agatha on consecutive pages. Nothing prevents you from even putting a heading page (just another document) labeled “Letters to Uncle Albert and Aunt Agatha” in front of them. If that is done, you have created, in effect, a file name but without having added a special mechanism in the software. You can create, if you wish, a hierarchical file structure of your own so that, if you really love file names and hierarchies, you can have them. Structure, as you choose to create it, is part of your *content*, not part of the *interface*.⁵ Instead of a file-finding facility, you simply use the general search mechanism to find the file names you have squirreled away. You can place a folder name as a document in front of a number of collected files; you can place a volume name in front of a bunch of collected folders. (Your pattern would be a volume character, followed by the name of the volume; such a pattern would eliminate matches to other occurrences of the name that happen not to be labels of volumes.) The absence of a built-in file organization does not prevent you from creating a file that fits your needs and one that, because *you* created it, you understand. But also, because the system has not been changed in any way, a guest can find things in your structure and, in fact, can ignore your structure and treat it as a flat, unstructured, file.

One advantage of filing information as you wish is that the structures were not dictated by the system designers, who may have ideas different from yours. Therefore, you do not have to develop a mental model of what the designers were trying to do. Many users do develop inaccurate models of how systems work; these mental models persist and cause continuing difficulties for those users (Norman 1988).

This discussion is not theoretical: On the SwyftWare and Canon Cat products, the elimination of file names, directories, and the various mechanisms usually provided for manipulating them proved one of their most

5. The same is true of file names and menus proposed in this book.

successful features. Users experienced at conventional computer systems sometimes found it difficult to make the transition to content-based organization, but once they had made the transition, the conventional methods soon began to seem cumbersome. Users who started on the Cat were not amused at having to learn the more complex and difficult methods of conventional file systems when they moved to a PC or a Macintosh.

To users accustomed to standard GUI practices, the methods outlined here may seem complex by comparison. But the apparent complexity is due to the new paradigm's lack of familiarity and to our having become habituated to the many steps we must take and to the problems we have with the current system. The learning curve has been long surmounted. But when you compare the progress of beginners with the two systems or compare the performance of experienced users, the advantages of the simpler system become apparent.

Consider that you have n documents that you want to copy to an external medium, such as a hard drive. With, for example, the Macintosh operating system (OS), you drag the icon of each document to the icon of the drive, and it is copied. In the new paradigm, it seems at first more complicated: You have to find the beginning and the end of each document, select the document, move the cursor to a place on the drive, and then move each document.

Recall that in the GUI, you start in the generating application. Your first step is to get to the desktop. You must also know which icons correspond to the desired documents, and you or someone else had to have gone through the steps of naming those documents. You also will have to know in which folder they are stored. So the apparent simplicity is arrived at only after considerable work has been done and the user has shouldered a number of mental burdens. A more efficient method involves an interface invention called LEAP. Assume, as we did for the GUI, that the cursor is in one of the documents you wish to move. With LEAP, the user can select the document with six keystrokes, and without having to look at the screen or recall the document's name. To type six keystrokes takes less time than to drag an icon.

LEAP works like this: There are two LEAP keys which lie under the thumbs. LEAP-Up searches forward and LEAP-Down searches backwards from the cursor position. Pressing and holding a LEAP key puts you into a quasimode during which whatever you type is used as a search pattern. To select the document you'd LEAP-Up to a document character (LEAP-Up↓ Doc↓↑↑). This puts the cursor at the beginning of the document. Then you'd LEAP-Down to a document character (the search will find the document character at the end of the document). A tap of both LEAP keys

together selects the text. (This is probably most easily done when the LEAP keys are operated by the thumbs, which are otherwise underutilized in typing. See Figure 2.1 for a typical keyboard designed for this use of LEAP. A dedicated Select key is another alternative.) In order that the function be visible, a legend adjacent to the LEAP keys is needed. The legend could, for example, read "Press both LEAP keys together to make a selection." Note that you do not have to watch the display while you select the document. Once a document is selected, the cursor is LEAPed to wherever you want to place the document. When the drive was plugged into the computer, its contents became part of the contents of the system; so no special mechanism is required to find it. A Copy command completes the operation. When selected this way, the document includes its separators. Thus, if the document is moved, it retains its character as a document because the document separators move with it.

The same technique used to copy a document—or a selection of any length from a character to a set of documents or the entire contents of the system!—from here to there is used to move a selection; the only difference is that a Move command rather than a Copy command is given. The process is functionally no more complex than that needed in the GUI. It is often faster, and the number of methods, concepts, and structures that an individual must understand is lower.

Consider how you would use the LEAP-based paradigm to put a few selections from different documents together onto a drive, and then consider how you would perform the same task using a GUI. With LEAP, the method is the same as that just described for moving documents to the drive: The selections are *found*—and once found, do not have to be opened, because the concept of opening a document is superfluous—*selected*, as described previously, except using the text rather than document characters at the beginning and end of the selection; and *copied* into place. In a GUI, the user must first *open* a new destination document, possibly by using the New command in the File menu of the application; *find* a document that contains a selection the user needs; *open* the document; *find* the selection within the document; *select* it; *use* the Copy command; *activate* the destination document; *paste* the selection in; *activate* the desktop; *find* the next document that has a desired selection; and *repeat* until all of the selections have been pasted into the destination document. Then you must *save* the result to the drive by using a dialog box.

Even if the complexity of doing any task was the same in either paradigm, the conceptual simplicity of the methods outlined here would be preferable. In most cases, the work required is also far less.

be achieved only on the rare occasions when the original interface has some major flaw that is remedied by a minor fix. (Parts of this section are based on Raskin 1994.)

6-2 Better Navigation: ZoomWorld

If you wanted to design a navigation scheme intended to confuse, you might begin by making the interface mazelike. The maze would put you in a little room with a number of doors leading this way and that. The doors' labels are usually short, cryptic, or iconic, and they may change or disappear,¹ depending on where you've been. You cannot see what is on the other side of a door except by going through it, and when you have gone through, you may or may not be able to see the room you've just left. There may not be a way to get directly back at all. Some rooms may contain maps to part or all of the system of rooms, but you have to keep track of the relationship between the map representation and the rooms you are presented with; furthermore, maps are not well suited to situations best represented by three-dimensional networks. The rooms in this description correspond to computer interface windows and web sites, and the doors are the tabs, menus, or links that are provided to bring you to other windows or sites.

As legends and stories from ancient times inform us, humans always have been notoriously bad at mazes. If we could handle them easily, they wouldn't be used as puzzles and traps. When using a complex program, I often find, deep in a submenu, a command or a check box that solves a problem I am having. When I run into the same problem a few weeks later, I cannot remember how I got to the box with the solution. We are not good at remembering long sequences of turnings, which is why mazes make good puzzles and why our present navigational schemes, used both within computers and on the web, often flummox the user. Many complaints about present systems are complaints about trying to navigate. Partial solutions, such as "favorite locations" in browsers, have been created.² But what we are truly better at is remembering landmarks and positional cues, traits that evolution has bred into us and traits we can take advantage of in interface design.

The antithesis of a maze is a situation in which you can see your goal and the path to get there, one that preserves your sense of location

1. Adaptive menus have this annoying trait.

2. This works until you have so many that you cannot remember what they all are; then you need a "favorites of favorites" or another scheme to keep track of them.

while under way, making it equally easy to get back. An elegant solution is the zooming interface paradigm (ZIP), which in many situations solves the navigation problem and also provides a way around the problem of the limited screen real estate that any real display system must confront. Imagine, if you will, how readily mazes could be solved if only you could fly above them, see their layout, and go directly to your destination. A zooming interface paradigm offers that kind of fluidity and facility for many tasks you perform with computers. Although a ZIP is not optimal for all situations, I will concentrate on its positive aspects, with the aim of demonstrating that there are alternatives superior to desktop-oriented graphical user interfaces.

The ZIP described here is called ZoomWorld and is based on the idea that you have access to an infinite plane of information having infinite resolution. The plane is ZoomWorld. Everything you can access is displayed somewhere on ZoomWorld, whether it is on your computer, on a local network to which your computer is attached, or on a network of networks, such as the Internet.

To see more of ZoomWorld, you think of yourself as flying higher and higher above it. To look at a particular item, you dive down to it. ZoomWorld also has a content searching mechanism. The overall metaphor is one of flying, climbing to zoom out and diving to zoom in. You navigate both by flying above ZoomWorld and by doing content searches.

ZoomWorld is conceptually similar to the walls of a project planning room. After a while, the walls become covered with annotations, tacked-up sheets of paper, sticky notes, photos, or whatever else helps us to remember and to explain our ideas. When you come into the room, you may stand in the center and glance around, spot a place, walk over, and look at the material in detail. At other times, you walk directly to the place where the information you want to review or modify has been posted.

We can find things in such a planning room because we tend to remember landmarks and relative position, a fact sometimes called the psi effect and long known to psychological researchers. "The stuff about marketing is on the right wall, sort of lower down near the far corner," someone might tell you. On another occasion, you go right to a particular document because you remember that it is just to the left of the orange piece of paper that Aviva put up. As you work in the room, you sometimes step back to get your bearings. You don't have to step back far before you can no longer read the small print, but you can see headings and larger diagrams. From still farther away, you can make out only a few of the largest headings, colors of whole sheets, layouts of areas, and tell whether there are illustrations,

diagrams, or cartoons on them, even if you can't make out just what is being illustrated or diagrammed.

The ZIP readily permits labels to be attached to images and to collections of images yet does not impose any structure, hierarchical or otherwise, beyond association due to proximity. I suspect that most users will tend to organize collections of images into clusters and that some will create informal hierarchies. For example, a large heading Personal Photos might, when zoomed in on, reveal smaller headings on groups of pictures labeled Baby Pictures, Vacations, Pets, Hobbies, Friends, Relatives, and so forth. Zooming in to the writing under the heading Baby Pictures might reveal the children's names, Agatha, Gideon, and Hermione. A professional photographer is likely to have a very carefully constructed and organized collection. Note that you are not required to remember the names; you find them as you zoom and glide about and decide whether the image you want is likely to be in the named category. The same can be done with films and collections of sound, although you may have to activate a film or a sound to make sure that it is the one you want.

The totally disorganized have hope: If you are one such, you can just lay out the images any old way and zoom around until you find the one you want. You may remember its approximate position due to your spatial memory, which will speed the search.

If you want larger characters, say, in a document you are reading, it is clear that zooming in makes the characters larger, but you may not be able to fit entire lines into the width of the display, making reading from the screen difficult. The solution in this case is to use a command to enlarge the font, making the line length, in terms of number of characters, shorter.

Our inherent tendency to remember position and landmarks can be put to work in ZoomWorld. The method of getting around will not be via scroll bars, which are slow. (Just consider, in Fitts' law terms (Section 4-4), the size of the arrow boxes you must click on.) Nor will it be by clicking on zoom-in and zoom-out icons or menu items, which are also slow, but rather by emulating what a person does in a planning room: stepping back to view large areas, then walking up to just the one that is desired, and finally leaning forward to read the fine print or using a magnifying glass to see a detail in a photo.

Zooming in, so that you can see portions of the space in detail, is probably best implemented as a quasimode (see Section 3-2-3). In this quasimode, the graphical input device button causes a real-time zoom-in to wherever the cursor is located: The center of zooming follows the cursor. The secondary graphical input device button can be used for this function.

For compatibility with the application-free systems described in Sections 5-7 and 5-8, the zoom quasimodes can be positioned elsewhere, such as on a second graphical input device, on the keyboard, or as dedicated, labeled extra buttons on the graphical input device (see Appendix A).

Wherever the zooming controls are located, the point you are zooming in on is the cursor location, which can be adjusted during the zoom by the graphical input device, being used in its role as a positional pointing device. That is, during zooming, the system moves the ZoomWorld plane so that the cursor position is at or near the center of the display. If zooming is quick—at least a factor of 2 in linear dimensions per second—and has the appearance of being continuous, zooming to and from the cursor position will suffice as a graphical navigational tool.

In the planning room, you might put up larger signs over the main areas, signs you can read from anywhere in the room. Zooming operates similarly; the sizes of headings and text determine how far in you have to zoom before you can see details. This technique substitutes for, and improves on, hierarchical directories. A fast text search, such as LEAP, is an important adjunct, used when you need to find a textual detail quickly. A few distinctive geometric landmarks can also be used; see the rules for effective use of icons in Section 6-3 for what is meant by "few." A large red cross, for example, might signal an area containing emergency medical data—in both the planning room and in a ZIP.

Without having the phenomenon pointed out to them, users quickly become familiar with the textures of the kinds of work they do. Spreadsheets, tables, text, bitmapped images, drawings, and other products of computer use each have their own visual characteristics even when seen from afar. The products of various coworkers, groups, vendors, and other creators of material will often be immediately recognizable.

A zooming space gives you a great deal of flexibility in layout. If a document grows in length or width, it can simply decrease its scale uniformly to make the entire document fit within the same region it did before its content increased. Similar space-making tactics can be used when a table or a spreadsheet is enlarged. Because you can zoom in, the text can always be made large enough to read. The opposite occurs when a document or drawing gets smaller. Blank space is unlimited, if the system is implemented properly; a command allows a new document to be started anywhere—by copying a blank document, the ZIP analog of the GUI New command—and documents can be allowed to overlap adjacent documents or be restricted in their growth by them. Internal links and pointers to web sites (URLs) can bring you immediately to other documents at a place and size

determined at the creation of the link. Buttons can have as much information on them as you wish, including a complete manual, with examples, without affecting their simplicity when viewed from afar. Every feature can have an explanation built into it.

Zooming can also be nonlinear in time, starting more slowly and accelerating to full zoom speed, allowing for fine control of small changes without impeding large changes in scale. It can also slow down and briefly stop at certain predefined zoom ratios, making it easy to get characters to standard sizes.

Working effectively in a ZIP often permits, and sometimes requires, strategies that are unlike those used in desktop-based GUIs. For example, you can have a number of views of the same data kept on hand because real estate poses no limitations. As with the efficient temperature converter solution in Section 4-3-2, we let the machine do extra work or preparation that may never be used, in the interest of easing human workload. Comparisons can be effected by a split screen, which requires controls for splitting but that allows for independent zooming, and by a typical ZIP trick: moving copies of the two documents or two copies of the same document so that the parts to be compared are adjacent. The ability to move and to scale any object certainly applies to text objects, which is how you can make a document bigger and thus visible at a higher hierarchical (and graphical) level.

A footnote can be more than just a reference. You can zoom in to the entire referenced work. Zooming here is functioning like a link, except that to get back to the main discussion, you zoom out; you do not have to keep a trail of where you have been. To make it easy to find a set of documents, the documents themselves can be arranged in a distinctive pattern that is visible when zoomed out. A page with very large lettering can be seen while zoomed out and used as a title. Zooming out can serve in the role of a Back button in a browser.

Small-scale collaborative endeavors are easily modeled by having collaborators all zoom on the same document, with appropriate rules to prevent interference. A network can be represented as a space in which everyone's work is placed in his or her own region. In a ZIP collaborative space, you can choose to keep as much of your work visible or invisible to others as you wish. Invisible documents are also a way to implement varying levels of access permission.³

3. An interesting zooming user interface (ZUI), called PAD++ (it is now called *Jazz*), has been developed independently, originally at the University of New Mexico. See <http://www.cs.umd.edu/hcil/pad++/>. I am grateful to Dr. Donald Norman, then at Apple, for pointing out this work to me.

A typical application for ZoomWorld arose in a design created for Apricus, which was trying to find a way to computerize a large (about a square meter) medical chart of the kind used in an intensive care unit (ICU). All of the methods tried proved slower than accessing the chart manually and also required excessive training; multiple displays could not be used to show the entire chart at once. Not only could ZoomWorld accommodate the chart, but also the same interface could serve as an enterprisewide database without significant extra development effort, allowing the company to extend its sights beyond automating the ICU database. The better interface allowed an expansion of the business model and the range of applications without requiring a more complex interface or any additional interface implementation, although a larger database was, of course, necessary. Figures 6.1-6.8 show versions of ZoomWorld that were developed at Apricus.⁴ In testing, it was found that nurses could use the system after less than one minute of training.

In Figure 6.1, the ZIP shows an ICU unit, surrounded by other units. You can zoom in to see the kinds of data available for any of the rooms, some of which are unoccupied. In Figure 6.2, you have zoomed in to room 132, and the major charts can be seen.

You can continue to zoom in to a particular chart. When the text becomes large enough to be readable, it also becomes editable. In Figure 6.3, a horizontal time scale and a vertical set of legends have automatically appeared. These "float" over the background image so that as you move around in the chart, the scales stay in place relative to the screen. They automatically disappear if you zoom out to or beyond the point where the text is unreadable or if you zoom inside an entry in the chart.

For example, you might zoom in to a particular value and find useful data, such as normal ranges for the measurement or even a lengthy quote from a medical text. Note that such detail does not get in the way or take up any apparent display space, yet it is there, where it is needed.

The zooming can take place in the other direction. Climbing high above Figure 6.1, our first view of the hospital, you can see that the ICU unit is on Floor 1, and you can tell what other units are on the same floor. In addition, the physical layout of the floor is apparent (Figure 6.4).

4. My thanks to Apricus for permission to describe its version of ZoomWorld and for the use of some of the screens as illustrations. Many details and the implementation were provided by Drs. David Moshal and Emanuel Noik and their crew. The customization to the hospital setting is due primarily to Betti Newburn, R.N.