Topics

Accessibility (9)
Advertising (9)
Agile (3)
Applications (33)
B2B Websites (5)
Content Strategy (27)
Corporate Websites (9)
Credibility & Persuasion (9)
Design Mistakes (15)
E-commerce (18)
Email (14)
Eyetracking (14)
Heuristic Evaluation (7)
Human Computer Interaction (68)
Information Architecture (12)
Interaction Design (5)
International Users (7)
Internet and Society (11)
Intranets (32)
Management (27)
Mobile & Tablet (46)
Navigation (38)
Non-Profit Websites (3)
Predictions & Milestones (36)
Prototyping (4)
Research Methods (41)
Search (10)
Social UX (9)
Standards (14)
Strategy (60)
Technology (47)
Trip Reports (20)
User Behavior (34)
User Testing (51)
Visual Design (13)
Web Usability (119)
Writing for the Web (53)
Young Users (3)

Jakob Nielsen's Alertbox: April 1, 1993

Noncommand User Interfaces

Summary: Several new user interface technologies and interaction principles seem to define a new generation of user interfaces that will move off the flat screen and into the physical world to some extent. Many of these next-generation interfaces will not have the user control the computer through commands, but will have the computer adapt the dialogue to the user's needs based on its inferences from observing the user. This article defines twelve dimensions across which future user interfaces may differ from the canonical window systems of today: User focus, the computer's role, interface control, syntax, object visibility, interaction stream, bandwidth, tracking feedback, interface locus, user programming, and software packaging.

Note: This is a revised version of a paper that appeared in the Communications of the ACM ***36****, 4 (April 1993), 83-99 [reference 95].*

Abstract

Several new user interface technologies and interaction principles seem to define a new generation of user interfaces that will move off the flat screen and into the physical world to some extent. Many of these next-generation interfaces will not have the user control the computer through commands, but will have the computer adapt the dialogue to the user's needs based on its inferences from observing the user. This article defines twelve dimensions across which future user interfaces may differ from the canonical window systems of today: User focus, the computer's role, interface control, syntax, object visibility, interaction stream, bandwidth, tracking feedback, interface locus, user programming, and software packaging.
Keywords: Agents, Animated icons, BITPICT, DWIM, Embedded help, Eye tracking, Generations of user interfaces, Gestural interfaces, Help systems, Home computing, Interactive fiction, Interface paradigms, Noncommand based user interfaces, Prototyping, Usability heuristics, Virtual realities, Wizard of Oz method.

1 Introduction

The backbone of most histories of computers is the simplistic model of "generations" of computers. We are normally considered to be in the fourth major generation of computers currently, with the fifth generation just around the corner. Most computer history textbooks use fundamental changes in the underlying hardware technology as the defining feature of computer generations, but as shown in Table 1, there are actually several other interesting dimensions of computing that have followed a set of generations roughly parallel with hardware developments, as the scope of user interfaces has broadened [47]. This article specifically deals with recent and coming changes in user interfaces that can be said to define or at least suggest the next generation of user interfaces.

Generation	Hardware technology	Operating mode	Programming languages	Terminal technology	User types	Advertising image
0 -1945 Pre-history	Mechanical, electromechanical (Babbage, Zuse Z3)	Not really being "used" except for calculations.	Moving cables around	Reading blinking lights and punch cards	The inventors themselves	None
1 1945-1955 Pioneer	Vacuum tubes, huge machines, much cooling, short mean time between failures.	One user at a time "owns" machine (but for a limited time only)	Machine language 001100111101	TTY, typewriter. Only used in the computer center.	Experts, pioneers	Computer as calculator

2 1955- 1965 Historical	Transistors; more reliable. Computers start seeing use outside the lab.	Batch ("computer as temple" to which you make offerings to get oracle replies)	Assembler ADD A,B	Line-oriented terminals ("glass-TTY")	Technocrats, professional computerists	Computer as information processor
3 1965- 1980 Traditional	Integrated circuits. Businesses can cost-justify buying computers for many needs.	Timesharing (online transaction processing systems)	"High-level" languages, Fortran, Pascal	Full screen terminals, alphanumeric characters only. Remote access common.	Specialized groups without computer knowledge (e.g. bank tellers)	Mechanization of white-collar labor
4 1980- 1995 Modern	VLSI. Individuals can buy their own personal computer	Single user personal computers	Problem oriented languages, spreadsheets	Graphical displays with fair resolution. Desktop workstations and heavy portables.	Business professionals, hobbyists	Personal productivity (computer as tool)
5 1996-? Future	Wafer-scale integration, computer-on-a-chip. Individuals can buy *many* computers.	Networked single user systems and embedded systems	Non-imperative, possibly graphical	"Dynabook" [61], multimedia I/O, easily portable, with cellular modem.	Everybody	Computer as entertainment

Table 1
Summary of the generations of computers and user interfaces. The generations are approximate, and not all dimensions d changed synchronously. The "advertising image" of the computer is from [3]. See also [130] for a table of four "paradigms of correspond to generations 2-5.

Most current user interfaces are fairly similar and belong to one of two common types: Either the traditional alphanumeric full-screen terminals with a keyboard and function keys, or the more modern WIMP workstations with windows, icons, menus, and a pointing device. In fact, most new user interfaces released after 1983 have been remarkably similar, and it is that category of canonical window system that is referred to as "current" in the rest of this article. In contrast, the next generation of user interfaces may move beyond the standard WIMP paradigm to involve elements like virtual realities, head-mounted displays, sound and speech, pen and gesture recognition, animation and multimedia, limited artificial intelligence, and highly portable computers with cellular or other wireless communication capabilities [44]. It is hard to envision the use of this hodgepodge of technologies in a single, unified user interface design, and indeed, it may be one of the defining characteristics of next-generation user interfaces that they abandon the principle of conforming to a canonical interface style and instead become more radically tailored to the requirements of individual tasks.

The fundamental technological trends leading to the emergence of several experimental and some commercial systems approaching next-generation capabilities certainly include the well-known phenomena that CPU speed, memory storage capacity, and communications bandwidth all increase exponentially with time, often doubling in as little as two years [6]. In a few years, personal computers will be so powerful that they will be able to support very fancy user interfaces, and these interfaces will also be necessary if we are to extend the use of computers to larger numbers than the mostly penetrated markets of office workers.

This article first considers basic ways of structuring the user's access to computational functionality, and then discusses the concept of noncommand-based user interfaces which seem to underlie several otherwise disparate developments in next-generation user interfaces. The article defines and surveys twelve dimensions along which next-generation user interfaces may differ from previous generations of user interfaces, and finally considers how the transition to next-generation interfaces may impact established usability principles.

2 Functionality Structuring

Traditional user interfaces were function-oriented; the user accessed whatever the system could do by specifying functions first and then their arguments. For example, to delete the file "foo" in a line-oriented system, the user would first issue the delete command in some way such as typing del, rm, zap, or whatever. The user would then further specify that the item to be deleted was called foo. The typical syntax for function-oriented interfaces was a verb-noun syntax such as "del foo."

In contrast, modern graphical user interfaces are object-oriented; the user first accesses the object of interest and then modifies it by operating upon it. There are several reasons for going with an object-oriented interface approach for graphical user interfaces. One is the desire to continuously depict the objects of interest to the user to allow direct manipulation [124]. Icons are good at depicting objects but often poor at depicting actions, leading objects to dominate the visual interface. Furthermore, the object-oriented approach implies the use of a noun-verb-syntax, where the file foo is deleted by first selecting the file foo and then issuing the delete command (for example by dragging it into the trash can). With this syntax, the computer has knowledge of the operand at the time where the user tries to select the operator, and it can therefore help the user select a function that is appropriate for that object by only showing valid commands in menus and such. This eliminates an entire category of syntax errors due to mismatches between operator and operand.

Unfortunately, the change from function-oriented interfaces to object-oriented ones is quite hard for interface designers. For example, in one recent study, five groups of developers with significant experience in the design of character-based interfaces were observed designing their first graphical user interface. Four of the five groups included function-oriented aspects in their design where object-oriented solutions would have been more appropriate, and the fifth group only avoided a function-oriented design due to advice from an outside usability specialist [100]. A follow-up study of one of the teams seven months later found that it had designed a good graphical user interface for a major product but that eight of the ten most severe usability problems in its prototype design were due to a lack of object-orientation. Object-oriented interface design is sometimes described as turning the interface inside-out when compared to function-oriented design, and this change is difficult for people who are used to the traditional way of structuring functionality.

An example may clarify the distinction between function- and object-oriented interfaces and show why it is not enough to be graphical in order to be object-oriented. Consider the task of selecting certain information from a database, formatting the data, and printing the resulting report. A function-oriented interface that was designed by participants in our study [100] started by asking the user to specify the query criteria in a (graphical) dialog box. Then, the user had to select formatting options from a (graphical) pull-down menu, and finally, the user could click on a (graphical) print button. Only after the last step would the user be shown any actual data from the database. All these steps were centered around the operations to be performed by the user and not around the actual data to be manipulated by the user. An alternative, object-oriented design would start by showing the user a window with sample records from the database. Observing this data would make it much easier for the user to remember the nature of the database contents, and would simplify the task of constructing an appropriate query. As the user modified the query, the system would dynamically update the content of the data window to show samples of records satisfying the query. Formatting would be done by modifying the window layout, thus providing immediate feedback on how typical records would look in the revised formatting. Issuing the print command would still be the final step, but the output would not be a surprise to the user, since it would only reflect the data-centered modifications for which incremental feedback had already been observed by the user.

The next generation of user interfaces will likely move somewhat away from the standard object-oriented approach to a user-oriented and task-oriented approach. Instead of using either a verb-noun or a noun-verb syntax, such interfaces will to some degree be syntax free. Gesture based interfaces [68][110] such as pen computing may feel like "paper-like" interfaces [144][146], and one certainly does not think of syntax when writing, drawing, or editing on a paper notepad. For example, allowing users to edit text by drawing proofreading marks on the text itself [76] eliminates the need for a syntax that distinguishes between separate indicators of what function should be executed and what object it should be applied to, since both are specified by a single proofreading mark such as, e.g., deleting a word by striking it out. The key notion here is that the specification of both action and object are unified into a single input token rather than requiring the composition of a stream of user input.

A further functionality access change is likely to occur on a macro level in the move from application-oriented to document-oriented systems. Traditional operating systems have been based on the notion of applications that were used by the user one at a time. Even window systems and other attempts at application integration typically forced the user to "be" in one application at a time, even though other applications were running in the background. Also, any given document or data file was only operated on by one application at a time. Some systems allow the construction of pipelines connecting multiple applications, but even these systems still basically have the applications act sequentially on the data.

The application model is constraining to users who have integrated tasks that require multiple applications to solve. Approaches to alleviate this mismatch in the past have included integrated software [99] and composite editors that could deal with multiple data types in a single document. No single program is likely to satisfy all computer users, however, no matter how tightly integrated it is, so other approaches have also been invented to break the application barrier. Cut-and-paste mechanisms have been available for several years to allow the inclusion of data from one application in a document belonging to another application. Recent systems even allow live links back to the original application such that changes in the original data can be reflected in the copy in the new document. However, these mechanisms are still constrained by the basic application model that require each document to belong to a specific application at any given time.

An alternative model is emerging in object-oriented operating systems where the basic object of interest is the user's document. Any given document can contain sub-objects of many different types, and the system will take care of activating the appropriate code to display, print, edit, or email these data types as required [5][25][40]. The main difference is that the user no longer needs to think in terms of running applications, since the data knows how to integrate the available functionality in the system. In some sense, such an object-oriented system is the ultimate composite editor, but the difference compared to traditional, tightly integrated multi-media editors is that the system is open and allows plug-and-play addition of new or upgraded functionality as the user desires without changing the rest of the system [15].

Even the document-oriented systems may not have broken sufficiently with the past to achieve a sufficient match with the users' task requirements. It is possible that the very notion of files and a file system is outdated and should be replaced with a generalized notion of an information space with interlinked information objects in a hypertext manner [90]. As personal computers get gigabyte harddisks, and additional terabytes become available over the network, users will need to access hundreds of thousands or even millions of information objects [30]. To cope with these masses of information, users will need to think of them in more flexible ways than simply as "files," and information retrieval facilities need to be made available on several different levels of granularity [38][108] to allow users to find and manipulate associations between their data. In addition to hypertext and information retrieval, research approaching this next-generation data paradigm includes the concept of piles of loosely structured information objects [78], the information workspace with multiple levels of information storage connected by animated computer graphics to induce a feeling of continuity [14], personal information management systems where information is organized according to the time it was accessed by the individual user [71][72], and the integration of fisheye hierarchical views [35] of an information space with feedback from user queries [70]. Also, several commercial products are already available to add full-text search capabilities to existing file systems, but these utility programs are typically not integrated with the general file user interface.

To conclude, several trends seem to indicate that the concept of files as uniform objects without semantic structure may be not continue to be the fundamental unit of information in future computer systems. Instead, user interfaces may be based on more flexible data objects that can be accessed by their content.

3 Noncommand-Based Interfaces

Several examples of next-generation interfaces can be characterized as noncommand-based dialogues. This term may be a somewhat negative way of characterizing a new form of interaction but the unifying concept does seem to be exactly the abandonment of the principle underlying all earlier interaction paradigms: that a dialogue has to be controlled by specific and precise commands issued by the user and processed and replied to by the computer. These new interfaces are often not even dialogues in the traditional meaning of the word, even though they obviously can be analyzed as having some dialogue content at some level since they do involve the exchange of information between a user and a computer.

Virtual reality [7][34][109] may be the ultimate example of a noncommand-based interface as it is based on immersing the user in a simulated world in which the user can move about in the same way as in a physical world [126]. For example, a virtual reality hockey game will allow users to play goalie by stretching their arms to place their hands in the way of the puck [134]. Of course, virtual reality systems may still include some traditional commands, like for example the use of a special gesture to materialize a menu of additional games. Even though a user interface may rely on noncommand interactions for some tasks, it is likely that there are many other tasks that are more naturally accomplished by explicit commands.

Certainly, noncommand-based interactions can take place with more limited hardware, even though the high-bandwidth devices allow more flexibility in matching interface expressiveness to the user's needs. For example, the card table system [112] allows two users to play card on two linked computer screens by dragging cards with their mouse. Each screen shows the cards in the local user's hand face up and the cards in the remote user's hand face down, and as cards are dragged onto the table, they are automatically flipped to be visible to both players. Both users can thus concentrate on the game and on moving the cards as they would in real life without having to issue specific commands to the computer. Note that this example is different from the canonical direct manipulation example of deleting a file by dragging its icon to the trash can, which is still a command-based interaction. The user only operates on the icon as a surrogate representation of the true object of interest (the document), and the user is thus in effect issuing a command to the system, since the user's actual intent is to remove the document from the disk and not to remove the icon from the window.

3.1 Eyetracking

Eye tracking has traditionally been considered an esoteric and very expensive technique, but recent eye trackers are becoming cheaper and more practical [54]. For example, some eye trackers observe the user by a video camera instead of requiring the user to wear special glasses. Users do not have full control over their eye movements and the eyes "run all the time" -- even when the user does not intend to have the computer do anything. Eye tracking is thus a potential input device for noncommand-based interfaces to the extent that the computer can figure out what the user means by looking at something. At a minimum, the computer can assume that users tend to look more at things they are interested in than at things they are not interested in, and this property has been exploited by systems like *The Little Prince* discussed below.

Since it is impossible to distinguish times when users mean something by a look from times where they are just looking around or are resting their gaze, one cannot use an eye tracker as a direct substitute for a traditional pointing device such as a mouse. Instead, special interaction techniques are needed. For example, it has proven possible to move an icon on the screen by selecting it by looking at it and then pressing a selection button (to prevent accidental selection), and finally looking where the icon should go [56][57].

Consider two different ways of controlling a paddleball video game, that is, a game with a sliding paddle that has to be positioned under user control so that a bouncing ball will bounce back off the paddle rather than fall through the bottom of the screen. The standard control for such a video game uses a direct manipulation interface in which the user moves a joystick in the direction where the user wants the paddle to go. The paddle keeps moving until the user returns the joystick to its neutral position. Alternatively, the paddle could be controlled by an eye tracker such that the paddle would be positioned at the x-coordinate of the location of the user's current gaze.

My experience from playing the eye tracker version was that I could just look at the screen where the ball was going, and get the paddle right under the ball with no real effort. This is a noncommand-based interface because I was not consciously controlling the paddle; I was looking at the ball, and the paddle automatically did what I wanted it to do as a side effect. In contrast, direct manipulation control involves some kind of command to explicitly move the paddle as such. The difference is one of the level of the dialogue: In eye tracking paddleball the user looks at the ball and the paddle keeps up by itself, whereas the user has to tell the computer to move the paddle left and right in a direct manipulation paddleball game. Therefore, the user's focus of attention remains on a higher and more task-oriented level in the eye tracking version of the game. Of course, the game may not be as fun when one can "cheat" by just looking at the ball, so an appropriate game design based on eye tracking might involve quite different types of games. This observation is a reflection of the fact that the user's real task in playing a game is to have fun and not to score as many points as possible.

Another example where the user's task is more traditional is an experimental naval display of ships on a map developed at the Naval Research Laboratory [56][57]. The screen contains a window showing a map of an ocean with icons for the ships of interest. There is also a window with more detailed information about the ships, and whenever the user looks from the map window to the information window, the information window is updated to contain information about the last ship the user had looked at on the map. This interaction technique is appropriate for eye tracking because no harm is done by updating the information window as the user looks around on the map. Therefore, it does not matter whether a look at a ship is intentional or not. The noncommand-based nature of this interface comes from the usage situation: The user goes back and forth between looking at the overview map and the detailed information, and always finds the relevant information without ever having to issue any explicit selection or retrieval commands.

My final example of a noncommand-based eye tracking system is an interactive fiction system called The Little Prince [125]. This system is based on the patterns of the user's eye movements aggregated over time instead of the individual movements. The application is a children's story based on the book The Little Prince. The computer screen shows a 3-D graphic model of the miniature planet where the Prince lives, and synthesized speech gives a continuous narration about the planet. As long as the user's pattern of eye movements indicates that the user is glancing about the screen in general, the story will be about the planet as a whole, but if the user starts to pay special attention to certain features on the planet, the story will go into more detail about those features. For example, if the user gazes back and forth between several staircases, the system will infer that the user is interested in staircases as a group and will talk about staircases. And if the user mostly looks at a particular staircase, the system will provide a story about that one staircase.

The point about The Little Prince from an interaction perspective is that the user never explicitly instructs the computer about what to say. In contrast to traditional hypertext systems [90], links to additional text are activated implicitly based on the computer's observations of the user and its conclusions about the user's probable interests.

3.2 Computer Music

Several systems have been built to allow the computer to provide accompaniment to music played by the user [13][21][52][133]. See also [111] for a survey of early work in this field. The basic principle of music accompaniment is that a small number of users (often only a single user) play their instruments in the way they normally would, and that the computer synthesizes the instruments that would normally be played by the rest of the orchestra. The computer provides appropriate accompaniment to the specific way the users play their instruments, based on its observations of the way the users are playing. These observations could in principle be made through a microphone and acoustic analysis, but are more commonly accomplished by special measurement devices attached to the user's instrument, since data from such instrumented instruments are much easier to analyze for underlying musical intent than are sound waves.

From a user interface perspective, some interesting attributes of music accompaniment are the use of untraditional input devices (flutes, pianos, violins, etc.) that are specialized for the users' tasks, the sound-based nature of the output, and the noncommand-based way the user controls the interaction.

In contrast, a computerized music synthesizer that follows the gestures of a human conductor as a live symphony orchestra would [81] is not a true noncommand system. Even though a conductor's commands are much higher-level than traditional programming and command languages, and even though gestures may be more natural than text for expressing musical intentions (especially the beat), the main point distinguishing the conductor interface from the accompaniment interfaces is that the user's focus of attention is to control the computer and instruct it to act in a certain way.

The interactive performance at the CHI'92 computer-human interface conference provided several examples of dancers generating computer music as a result of their movements, which thus served as "input devices" to the system. Leslie-Ann Coles was observed by a video camera doing gesture recognition, Chris Van Raalte was wired with electrodes on his skin to sense muscle contractions, and Derique McGee wore a data-suit that could sense when he slapped his body. All three dancers demonstrated not just untraditional input devices, but also the use of the entire stage as interactive space, thus liberating the computer interface from being tied to the workstation.

3.3 Agents

Interface agents are another approach to alleviating the user of the burden of having to explicitly command the computer [73][75], and have even been called "the next big direction in user interface design" by a respected user interface expert [60]. Agents are autonomous processes in the computer that act on behalf of the user in some specified role. The eventual goal of some researchers is to have highly intelligent agents that know the user's schedule, can retrieve exactly the desired information at any given time, and in general combine the functions of butler and secretary [27]. For example, a very effective demo of the conversational desktop system [119][120] had the computer remind the user of a scheduled flight (possible through knowledge of the user's calendar), that traffic to the airport currently was heavy (possible through a link to the city's traffic computer), and offering to call a cab (possible through speech synthesis or a direct computer link to the taxi company). Other goals of agents are to enable the computer to live up to the Japanese saying "know 100% by hearing 10%" [102] since the agent would have a model of the user's work habits and would thus not have to be instructed in detail.

Agents can also be very simple. For example, an agent might count the number of times a user gives an invalid command and then offer the user an explanation when the count reaches a certain number.

Even without the high level of artificial intelligence and the excessive requirements for standardization of information exchange needed to support some of the more fancy scenarios, agents can still help users with many tasks. For example, Object Lens [69] allowed users to construct agents to sort and filter their incoming electronic mail according to various criteria. A typical agent could search for talk announcements and place them in a special mail folder from which they could be automatically deleted after the announced date of the talk unless the user had moved them to a permanent archive first.

Agents allow the computer to take the initiative to initiate interactions with the user. A typical example is the proposal for activist help systems [33]. Traditional help systems are all passive in the sense that they do not offer help unless the user explicitly asks for it. This has the obvious problem of being one more thing to do (and possibly to do wrong). Also, users do not always know when they might benefit from asking for help. In contrast, activist help systems use an agent to monitor the user's interactions with the system. If the agent senses that the user is in trouble or is using the system in an inappropriate manner, it can decide on its own to initiate a help dialogue and offer help to the user.

3.4 Embedded Help

Even though activist help may solve some of the problems with traditional passive help, there is a major risk that users will find the computer intrusive and nagging if it interrupts their work too frequently with advice. Also, activist help is still at heart a separate help system and thus still adds to the user's overhead in using the computer. Alternatively, embedded help is an approach where help is integrated with the user's primary task environment and made available as needed without any explicit user actions. An example of embedded help is the way building directories are often found in elevators -- sometimes even integrated with the buttons or the floor indicator (the elevator's user interface).

Some computers already offer a simple form of embedded help in the form of context sensitive help messages that appear whenever the user touches specific parts of the interface. Such context sensitive help can take the form of an extended cursor with explanations of the function of each of the buttons on a multi-button mouse [1][82], pop-up annotations with short descriptions of icons, menus, or dialog box elements being pointed to [122], or even the line at the bottom of the screen used by some systems to preview or summarize the result of choosing each option as the user moves through a menu.

Experimental help systems have demonstrated the use of knowledge about the users' data objects to generate tailored animations showing how the users' would manipulate their own data to achieved a desired result [127]. Superimposing help animations over the regular screen display of the interface provides a tighter level of integration than traditional help systems that show help information in separate windows. Such highly context-sensitive help approaches the embedded help ideal, but normally still requires an explicit user command to be activated and also maintains some distinction between the help system (where user actions may be animated, but cannot be carried out) and the "real" interface.

Next-generation interfaces may provide true embedded help by the use of animated and auditory icons, thus employing their multi-media capabilities to make the computer easier to use and not just prettier to look at. Animated icons [4] show looping animations that some times illustrate the meaning of an icon better than a static image. For example, many paint programs use an icon of an eraser to indicate the erase function. Experience has shown, however, that novice users often do not recognize such icons as being pictures of erasers. Some users think that the icon represents a feature for drawing boxes on the screen, and others are simply mystified. Erasing is a difficult concept to illustrate in a static picture because it is a dynamic process rather than a concrete object or attribute. In contrast, an animated icon could show an eraser being moved across a patterned surface and how the pattern had been erased in the path of the movement.

A screen filled with several constantly animating icons would probably be distracting to users. Alternatively, animated icons can be designed to only animate when the user indicates an increased level of interest in them, for example by placing the mouse cursor within such an icon. By only animating the icons when they are being pointed at, the animations serve the role of embedded help.

Auditory icons [41] are characteristic sound effects that are played to provide additional information about user actions or system states [42]. For example, when the user clicks on a file icon, the computer can play different sounds as feedback, depending on what type of file is being selected. Computer-generated files could be assigned, e.g., more metallic sounds than user-generated files. When the user deletes a file by putting it in the trash can, the computer can play a dramatic crashing sound if the file was large, and a puny sound if the file was small, just as physical trash cans sound different, depending on what is thrown away. The reason this example is a kind of embedded help is that the user's natural actions can be made to reveal additional information about the system without interfering with those actions. Hearing a small sound whenever files are discarded assists the novice user in understanding what is going on, but soon becomes an expected part of the interface. The auditory icons reemerge as a helping part of the interface in cases where users trash large files, believing they only contained, say, a few short notes. The mismatching sound will startle the users and cause them to retrieve the files for a closer examination. The point here is that interface elements like sound can add to the richness of the dialogue and thus provide additional cues to the user without adding to the complexity of the primary interaction. Alternative interface techniques for providing additional information, such as a dialog box to confirm deletion, add overhead by necessitating explicit user action. Also, they require the user to pay conscious attention to them if they are to be of any help, whereas auditory icons remain in the background as long as no exceptional cases are encountered.

4 Interaction Characteristics for Next-Generation Interfaces

Table 2 summarizes twelve dimensions along which next-generation user interfaces may be different from traditional interfaces. These dimensions are discussed below. One should probably not expect all next-generation user interfaces to differ from current user interfaces on all of these dimensions simultaneously, but many systems will probably include changes on more than one dimension. It seems that the design trends listed as being "next-generation" tend to support each other along several of the dimensions.

	Current Interface Generation	Next-Generation Interfaces
User focus	Controlling computer	Controlling task domain
Computer's role	Obeying orders literally	Interpreting user actions and doing what it deems appropriate
Interface control	By user (i.e. interface is explicitly made visible)	By computer (since user does not worry about the interface as such)
Syntax	Object-Action composites	None (no composites since single user token constitutes an interaction unit)

Object visibility	Essential for the use of direct manipulation	Some objects may be implicit and hidden
Interaction stream	Single device at a time	Parallel streams from multiple devices
Bandwidth	Low (keyboard) to fairly low (mouse)	High to very high (virtual realities)
Tracking feedback	Possible on lexical level	Needs deep knowledge of object semantics
Turn-taking	Yes; user and computer wait for each other	No; user and computer both keep going
Interface locus	Workstation screen, mouse, and keyboard	Embedded in user's environment, including entire room and building
User programming	Imperative and poorly structured macro languages	Programming-by-demonstration and non-imperative, graphical languages
Software packaging	Monolithic applications	Plug-and-play modules

Table 2
Comparison between the current user interface generation of command-based interfaces and the potential next generation of interfaces across twelve dimensions.

4.1 User Focus

Users have traditionally been required to pay close attention to the control of their computer system, to the extent that the use of a computer often feels like exactly that: the use of a computer, and not like working directly on some task. Users have been required to come up with the appropriate commands and to put together command specifications in the appropriate syntax. These requirements are typically completely overwhelming for novice users, and even experienced users often have no real desire to excel in using a computer as such but would like to be able to concentrate on doing their work [16][99].

As mentioned in the previous section, many next-generation user interfaces seem to be based on some form of noncommand interaction principles in order to allow users to focus on the task rather than on operating the computer. Some systems may be as specialized as appliances and take on a single role without further need for user instruction.

For example, the Portholes system for connecting work groups at remote locations displays miniature images of each participant's office as well as meeting areas [24]. These images are refreshed every few minutes and thus allow people at each location to get a general idea of which colleagues are around and what they are doing, but without the privacy intrusion that might follow from broadcasting live video. For the purposes of the current discussion, an important point about Portholes is that the various participants do not need to take any action to inform their co-workers that they are in their office or that they are meeting with somebody and should not be disturbed. This information is communicated to the system by virtue of the regular activities they would do anyway, thus allowing them to focus on their real-world task and not on using a computer. Experience with other systems for computer-supported cooperative work has shown that people are reluctant to expend effort on entering information into a computer for the sole purpose of helping others [46], so this type of interface design to allow users to focus on their work is probably the only one that would work in the long term.

4.2 Computer's Role

Many users would probably prefer a computer that did what they actually wanted rather than what they said they wanted, but traditional computer systems explicitly follow a command-oriented interaction style where the computer does exactly as it is told, even when the user's commands may be different from the user's intentions. One of the few exceptions from this rule was the DWIM (Do What I Mean) feature [129] of the Interlisp programming system [117]. In DWIM, the computer would reinterpret meaningless user input to make it into legal commands. For example, if the user issued a command referring to a non-existent file, DWIM would not issue an error message but try a spelling correction on the file name. If a small change in the file name made the command legal, DWIM would assume that the user had mistyped, and it would reissue the command in a corrected form. In control rooms or other systems where the computer has ways of assessing whether the user is about to make a mistake, it can also be possible for the computer to interrupt users and warn them against the likely consequences of their actions [53]. A similar example is the Lisp Tutor [2] which interrupts the student's programming efforts if it knows that the student has made a programming error in a sample problem. By not waiting for the student to finish the program and run it, the tutor can provide instructional material at the point in time where the error is fresh in the student's mind.

Many of the examples discussed above illustrate ways for the computer to watch the user and interpret the user's actions (or even inaction, which might indicate a need for help under certain circumstances) [10]. Such inferences are becoming more feasible as the computer gains additional high-bandwidth media through which to observe the user, as discussed below. Not only may the user be observed by eye-tracking and special equipment such as datasuits and active badges (discussed further below), but the computer may also point video cameras at users to get a general idea of where they are.

Even though some forms of agents can be implemented without the use of artificial intelligence [116], the use of agents in the interface will probably be most successful if they can rely on some form of limited artificial intelligence in the system. This does not mean, however, that it will be necessary to wait until full artificial intelligence is achieved and perfect natural language understanding becomes possible. Several of the interface techniques discussed in this article require the computer to make some kind of semi-intelligent inferences, to build up knowledge-based models of the users and their tasks, and to perform fairly complex pattern recognition. It is not necessary, though, for the computer to fully understand the domain or to exhibit human-like qualities in other ways. The interaction is still that: an interaction between two participants; and the human can supplement the computer's limited intelligence.

4.3 Interface Control

It follows from the above discussion of the changes in the user's and the computer's role in the interaction, that much of the control of the user interface will pass from the user to the computer. Sometimes, the computer may even choose to perform actions without explicit user control, and often, it will customize the interaction by changing appropriate parameters automatically.

When the computer is allowed to change the user interface, it can adapt the interaction to the user's specific usage circumstances and location. For example, if the computer knows where the user is, it can enlarge the text on the display if the user is standing up, or it could speak out important alert messages by speech synthesis if the user was in the other end of the office. Furthermore, the computer could act on important electronic mail arriving while the user was out of the office by one of several means: activating the user's beeper, ringing a phone in the office where the user was, downloading the message to the user's notebook computer over the wireless network, or sending a fax to the user's hotel. The exact choice of delivery mechanism would be chosen by the computer based on knowledge of the user's whereabouts and preferences.

Computer control of the interface may be resented by some users if it is not designed carefully. Many forms of adaptive interfaces may be readily accepted because they simply cause the computer to behave the way one would naturally expect it to do if it were part of traditional physical world. For example, the organization of kitchen tools in drawers and cabinets adapts by itself to cause the most frequently used tools to be on top and in front, whereas less frequently used tools are hidden [51]. In a similar manner, several current applications augment their "File" menu with lists of the last five or so files used by the user in that application, under the assumption that recently used files are likely to be among the more frequently used ones in the future and thus should be made more easily accessible. This assumption seems reasonable, and a study of somewhat similar adaptive menus found them to be an improvement over static menus [45]. Given the observation that users tend to have several working sets of data and tools that are used together [49], it might be better, though, to have the computer build cross-application object lists that can be associated with the user's various tasks. In other words, following the above-mentioned trend away from monolithic applications, adaptive stand-alone applications may not be sufficient to meet the user's needs, and the computer may have to build a system-wide model of the user's work across application objects.

4.4 Syntax

As mentioned in an earlier section, syntax considered as temporal rules for the sequence of input actions may disappear or at least be greatly diminished in importance in many next-generation user interfaces. Syntax was necessary in earlier interfaces because they relied on a limited user vocabulary that had to be combined in order to specify complex actions. In contrast, gesture-based interfaces provide almost infinite numbers of unique input tokens that can specify a complete unit of intent, given that the location of the gesture is also significant for its interpretation. Furthermore, increased use of multiple parallel input streams (discussed further below) and the elimination of explicit turn-taking in the dialogue (also discussed below) make it possible and necessary for the computer to be more flexible in handling many alternate sequences of input actions, thus reducing the incidence of syntax errors.

At the same time, gestural languages may introduce a visual syntax for more complex operations to supplement the role normally played by temporal syntax [143]. For example, the parsing of gestures corresponding to this expression

$$\sqrt{\boxed{\frac{x+y}{2}}}$$

certainly needs a visual syntax to determine that the scope of the square root sign is that denoted by the bounding box (which would of course not normally be visible to the user) and not just the "x" [48].

From a usability perspective, the square root example may still feel syntax-less, since the user only needs to draw a single gesture to specify both the operation and its scope. A key advantage here is the transparency of the underlying parsing due to the user's understanding of two-dimensional mathematical notation. In reality, such understanding is initially quite difficult to achieve, and the usability of a gesture-based formula editor could be very low for users who had not already internalized the relevant composition rules. Actually, most of the interaction techniques discussed in this article are only "natural" for users who are already used to the basic elements of the techniques. The advantage of next-generation interfaces is that they tend to build on abilities that many humans have historically acquired anyway (such as looking at the world, handwriting, and gesturing), such that the learning time for these skills is not charged to the computer.

4.5 Object Visibility

Direct manipulation [124] is a fundamental component of most current graphical user interfaces, with the prototypical examples being the way icons can be deleted by being dragged to the trash can, and word processor margins can be adjusted by the dragging of markers in a ruler. Direct manipulation almost by definition requires that the objects that are to be directly manipulated are made explicit to the user and represented visibly on the screen.

It is not possible to directly manipulate a file system with millions of data objects that all have to be visible at the same time. Instead, some objects will be hidden or manipulated implicitly through agents or as side-effects of other user actions. Reduced object visibility may inversely impact usability unless care is taken to allow the users ways to find objects and inspect their state as needed.

4.6 Interaction Stream

The Toronto interaction device specialist Bill Buxton sometimes jokes that mouse-based user interfaces seem designed for Napoléon, in that they in principle would allow the user to keep one hand under the coat. In general, current user interfaces are based on single-threaded dialogues where users operate one input device at a time. For example, the user can either be using the mouse or the keyboard, but not both at the same time. The main current exception is the use of modifier keys like Shift-Click or Option-Click, but such modifiers are essentially mouse actions that could as well have been supported by having a few more buttons on the mouse.

In contrast, future interfaces may involve multi-threaded dialogues, where the user operates multiple input devices simultaneously to control different aspects of the interface [50][128]. Buxton and Myers [12] showed that users were able to use both hands in parallel to operate a traditional command-based interface controlled by a graphics tablet for one hand and a set of sliders or a touch sensitive surface for the other hand. Users who were allowed to use both hands in a multithreaded dialogue performed a test task 15-25% faster than control groups who were constrained to using one hand at a time.

The handling of multithreaded input will obviously also become necessary if the computer is to be able to observe the user by some combination of eye tracking, video cameras, active badges, and so on. An example of multithreaded input is the classic Put-That-There system [9], where the user could move objects by pointing at a wall-sized display and say, "put that" (pointing to an object) "there" (pointing to a destination). The system combines gesture and speech recognition and requires both to run in parallel, since the recognition of the screen coordinates being pointed at has to take place at the exact time the user says "that" or "there." Either user action is meaningless without the other, so the computer cannot complete, say, the analysis of the speech input before paying attention to the gesture tracker.

Examples of multithreaded output obviously include the many multimedia systems that have appeared recently. Also, some Help systems use the audio channel to comment on events on the graphical screen without changing or interfering with the display. One experimental system used a virtual reality-type interface for "parking" additional windows and icons in a simulated space around the user's primary workstation [31]. Objects in the simulated space were made visible through a see-through head-mounted display which was combined with a head tracker to determine what objects to display, depending on where the user was looking. The see-through display had fairly poor resolution, so users would move their primary working windows back to the real computer screen and only use the simulated space for currently unused windows.

Multithreaded input and output has several advantages from a user interface perspective. First, as in the Put-That-There example, the different input media may supplement each other. Sometimes, one medium can be used for one stream of input, such as commands, and another can be used for another stream, such as data, with the resulting dialogue feeling less constrained and moded than when both streams have to be overloaded on a single device such as a mouse. For example, a drawing program could use pen input for the graphics and voice recognition for commands like undo, rotate, etc. As another example, a low-bandwidth input mechanism such as a foot pedal might be used to denote temporary mode-changes, thus making the modes less confusing to the users because of their spring-loaded nature [123]. A second advantage of combining multiple input streams is that they will often allow more precise recognition due to the redundancy exhibited by natural human behavior [87]: if the computer has difficulty understanding what the user is saying, then knowledge of where the user is looking may help decide between two possible interpretations if one of them matches the object the user is currently looking at.

4.7 Bandwidth

The user's input to current systems has either very low bandwidth (a keyboard generating maybe ten characters per second), or at most a fairly low bandwidth when the user is moving the mouse. Even the output to modern graphics displays effectively has a fairly low bandwidth since most of the pixels on the screen remain the same for several seconds at a time as long as the interface is based on static images.

In contrast, the various next-generation user interfaces described in this article demand significantly increased bandwidth between the computer and the user. Tracking the motion of a dancer's body in three dimensions in order to generate appropriate music requires several orders of magnitude more communication and recognition capacity than the tracking of a mouse in two dimensions. Likewise, the generation of stereoscopic animated graphics with sound effects for a virtual reality requires much higher bandwidth than the displaying of a dialog box. Studies of virtual reality systems have shown that users notice time lags of as little as 200 milliseconds between their motions and updates of the headmounted display [104].

4.8 Tracking Feedback

Providing users with feedback during the dialogue is one of the most basic usability principles [96], and continuous feedback offers users the possibility of adjusting their actions before they have committed to an erroneous result. Traditional, direct manipulation graphical user interfaces are often good at providing continuous feedback based on the lexical level of the dialogue. For example, as the user moves a file icon over an application icon, the computer will highlight the application icon if the application knows how to open a file of the type designated by the file icon. This kind of feedback only relies on lexical knowledge of the identity of the basic interaction tokens (here, the types of the icons) but not on deeper knowledge of the user's intentions or the semantics of the interaction objects. It is therefore possible to provide the highlighting feedback by a low-level process that tracks the mouse motions and tests the types of the icons touched by the pointer.

Tracking feedback may be much more difficult to achieve in some next-generation interfaces. For example, a music accompaniment system cannot always generate the same sound as a result of a given note played by the user. The appropriate feedback depends on the type of music being played and the context of the user's other input. Also, systems relying on the recognition of free-form user input, such as a natural language speech recognizer, probably need to provide continuous feedback on the way the user's input is being interpreted, such that the user can rephrase the input if necessary. One fairly unobtrusive way of doing so in an agent-based system is to have an anthropomorphic visualization of the agent's understanding in the form of nodding when it understands [136] and frowning when is doesn't [28].

Gesture-based interfaces present special tracking problems in that appropriate user feedback often cannot be given until after the gestures have been recognized, meaning that the feedback will appear too late to help the user in completing the action. One suggestion for alleviating this problem is to design hybrid gesture-direct manipulation interaction techniques [115] where tracking feedback appears halfway through a gesture (which may even be recognized early through "eager" recognition). Alternatively, one might design interfaces where progressively improved tracking feedback appears throughout the user's action, as more of it is recognized.

4.9 Turn-Taking

Traditional user interfaces have been based on the concept of a dialogue where the computer and the user took turns in presenting statements to the other dialogue partner. While the computer was waiting for the user's input, it would sit idle, and the user was also prevented from initiating new actions during any response time delays, with the possible exception of typeahead which was not processed anyway until the user's proper "turn." A typical example of turn-taking is the way database searches and information retrieval have alternated between the specification of user queries and the display of the returned set of information, leading to fairly slow progress towards iteratively focusing on the desired information. Dynamic queries where the system works in parallel with the user without waiting for the user to finish specifying a query were 52% faster than a traditional database in one test [142] and were much preferred by users.

The granularity of the turn-taking in dialogues has been getting steadily smaller from hours or days in the batch processing era to seconds or minutes in the full screen form fill-in days, to sub-second interaction units in modern graphical user interfaces. Except for video games, the principle remains, though, that first the user commands, and then the computer replies. Many graphical user interfaces are implemented as event-based programs, and some of them take advantage of this software structure to allow the user to continue to interact with the program while earlier, time consuming commands are still in the process of being carried out.

Many next-generation interfaces will abandon turn-taking because they will have no well-defined transition points where the user would stop and wait for a response. This is typically true for noncommand-based systems as discussed above. For example, the VIDEOPLACE system [65] projects a silhouette of the user onto a large screen where it is merged with images of a simulated world. The effect is that of stepping into the images and is similar to virtual realities, but in 2-D rather than 3-D. Using VIDEOPLACE involves playing with simulated critters crawling over you or with the silhouettes of other users, as well as picking up and stretching various objects. All of these activities are continuous, and users just keep playing without any specific system response, except of course from the fact that the system continuously keeps up with their activities and generates new critters and objects for them to play with.

Some next-generation interfaces will not only allow the user to provide simultaneous input across multiple channels as discussed above, but will also keep providing new output throughout the user's input actions. In systems that are in some way coupled to the physical world, the need for continuous system output is obvious in that the world does not wait for the user before it changes its state. So systems for, say, air traffic control or the monitoring of a telephone network will have to update their displays to reflect changes in the surrounding reality. Similarly, systems for giving directions to a driver (discussed further below) will keep monitoring the progress of the car on the route and will have to interrupt the driver when an important turn is reached even if the driver was in the middle of issuing a command to the system.

4.10 Interface Locus

Users of traditional computers have been chained to their screens to the extent that many non-technical users talk about the screen as if it were the computer. Binding the interface to the screen of a workstation or a terminal has severe limitations for the use of the computer, however. There are obviously many human tasks that are not done while sitting at a desk and that can only be supported by computers with less desk-bound interfaces. Activities such as those of travelling salespeople may be helped by the current trend towards pen-based, highly portable computers [107]. Other human activities involve the collaboration of many people sitting in meetings or walking about a plant, construction area, or other non-office environment, and these activities may be helped by a trend towards making computational power available as part of the environment rather than limiting it to the flat screen [138]. Wireless networks will allow users to carry smaller computers with them without losing touch with their main computer, and in fact, the distribution of data and functionality among different computers may become transparent to users who will feel that all computational devices are access points to "their" computer, no matter where it is physically located, and no matter whether the data being accessed is in fact on their personal computer or on a remote host.

Plain usability considerations also support some moves of the interface into the environment. For example, virtual reality interfaces may be more convenient to use when they are projected onto the walls of a media room [19] instead of requiring the user to wear a special head-mounted display. Similarly, eye tracking, voice input [132], and some gesture-recognition interfaces are more pleasant to use if they allow the user to move about rather than sit in a fixed position all day. Also, computers may project information onto physical objects instead of being restricted to a separate display screen [32][140].

Initial moves away from the flat screen include the use of non-traditional input-output devices that have the feel of physical objects in their own right and not just as appendices to a computer. A prime example was the Noobie "playstation" [26] which had the shape of a large fantasy animal. The user (typically a child) would interact with Noobie by sitting in its lap, squeezing its tail, or moving its arms. Noobie's output was a traditional computer screen that would display various other fantasy animals in accordance with the user's manipulation of Noobie's body. Empirical studies of children using Noobie showed that they did not wonder where the keyboard or mouse were, but readily accepted that a furry creature could be an interactive device.

Output devices may also become embedded in the environment and interact with characteristics of that environment. For example, active employee badges [135] that constantly transmit the identity and location of every individual in a building have been used to support a personalized corporate bulletin board displaying information of interest to those employees who happen to be passing by it [137]. Note how the user's only "command" to the system is his or her physical presence.

Real-world objects serve as the total user interface to the experimental DigitalDesk system [88], where the user's regular desk is observed by the computer through a camera mounted in the ceiling. When the user gestures to characters on a piece of paper in a special way, the computer performs optical character recognition on the camera image of the paper and acts on the information. Output can be displayed on the same paper from a projector mounted next to the camera. For example, the user could gesture at a column of numbers in an expense report to have the system calculate the sum and project the result at the bottom of the column [139]. As another example, a user reading a foreign language text could point to a word and get the dictionary definition displayed, thus making any printed book into a kind of hypertext as long as it was in view of the camera and projector of the system.

Other examples of integrating the surrounding environment with the user interface include mobile systems like computer-equipped shopping carts that display appropriate advertisements depending on where the user goes in the supermarket (and thus which product groups the user is interested in). In fact, the proper term for the human in this human-computer interface may be "shopper" and not "user," illustrating how the computer blends in with the environment and allows the human to "use" it while remaining focused on other tasks. Though such systems are currently being deployed in supermarkets solely for advertising reasons, one could easily imagine them integrated with the user's home computer from which the shopping cart could download the user's shopping list for the day and steer the user towards the location of the desired items [64], possibly using information retrieval techniques like latent semantic indexing [23] to match the user's vocabulary to database records describing the goods in the store.

The experimental Back Seat Driver system [22][121] uses speech output to provide directions to a driver of a car navigating city streets. The system can also deliver voice mail and weather reports, but its main responsibility is to get the driver to the destination by giving instructions just as they are needed. Instead of traditional written directions like "turn right at the fifth traffic light after the square," the Back Seat Driver can give instructions like "turn right at the next traffic sign," and even "you just missed your turn" (after which it can plan an alternate route). The prototype location system determines the car's current location by dead reckoning from a known starting point, but one could also imagine using navigational satellites or other location systems. For the purposes of the current analysis, important attributes of the Back Seat Driver are that the user's "input" to the system simply consists of driving a car, that the system is located where the user needs it and not in a special box, and that its output is determined relative to the user's current location.

4.11 User Programming

End-user programming of current user interfaces mostly involves a profusion of very awkward macro languages that in many cases do not seem to have benefited from advances in programming language design in the last thirty years. In fact, end-user programming has declined in usability with the introduction of graphical user interfaces which increased the gap between the normal representation of the interface and the textual encoding needed for current scripting languages. Also, as the increased general usability of computers allows users to learn (and thus use) a larger number of different software packages, the inconsistency in having separate scripting languages for each application has become more of a problem. The solution to the inconsistency problem is obviously to designate scripting as a system-level service that can apply to all applications. System-wide user scripting may be easier to achieve in coming object-oriented operating systems, but some advances are already being made in building application programmer interfaces that can react appropriately to events generated by other applications, including scripting facilities.

In spite of the poor quality of many current macro and script languages, they are widely used for tasks ranging from the building of custom spreadsheet applications to the customization of individual users' environments [59], indicating the need for end-user programming. Approaches to making end-user programming easier include the introduction of object-oriented ideas to allow inheritance and specialization [58], for example allowing users to build up customized environments through gradual changes and copying of other users' programs that are made explicit on the screen as buttons [77]. Also, it is likely that next-generation languages for end-users will be graphical or at least include graphical elements [84] to minimize the mismatch between programs and experienced interactions. For example, BITPICT [36][37] is a rule-based graphical programming language as shown in Figure 1, that allows users to request changes in a graphical environment by specifying the way interface elements looked before and after the change. As another example, editable graphical histories [66] allow users to manipulate previous system states through a comic-strip metaphor [67].

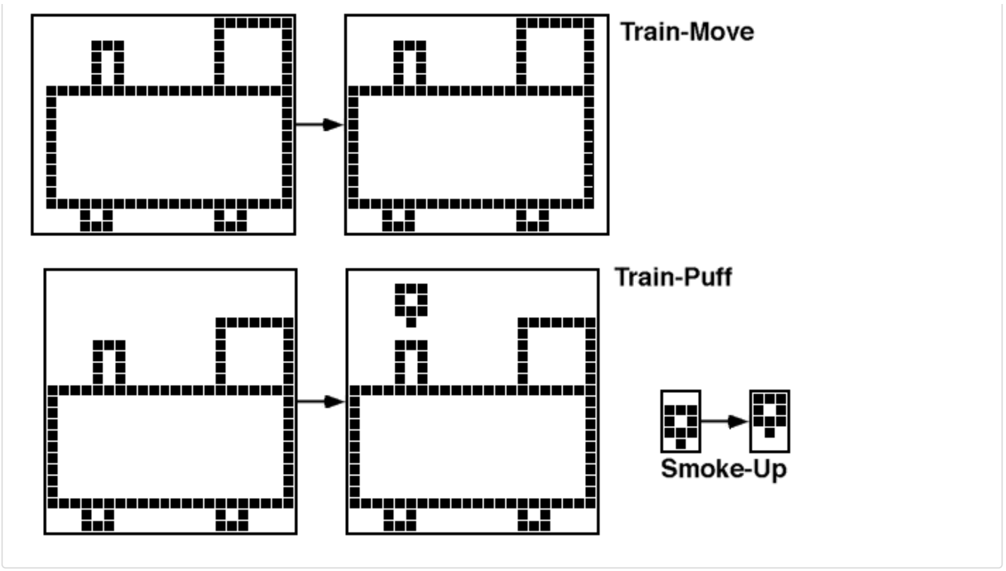

Figure 1
Example BITPICT program. These production rules implement an animation of a steam train. The Train-Move rule causes the train to move left, one pixel at a time as long as there is white space to the left of the train. Train-Puff causes smoke to appear from the smokestack when it has been cleared of previous smoke (note the many white pixels above the train), and the Smoke-Up rule causes the smoke to rise one pixel at a time. This program was generated by the author the first time he used BITPICT.

There is a conceptual conflict between the desire for end-user programming and the trend towards noncommand interfaces as discussed above. One development that is likely to alleviate the user of much of the burden of generating program code is programming by demonstration [83][85][86]. The basic principle is that the user enacts examples of the behaviors that need to be automated and lets the computer write an appropriate program to cause such activities in the future.

In some programming-by-example systems, users may need to go into a special demonstration mode when they want to construct a new program. Other systems allow users to continue focusing on their work and allocates the responsibility also for the programming aspect of the interface to the computer, thus following two additional next-generation principles from Table 2. An example is Eager [20] which automatically constructs macros for repetitive tasks based on observing the user, identifying repeated actions, and making inferences about how to automate such actions. For example, a user might decide to copy the subject fields from a set of email messages to a single overview file. At first, the user manually copies a field, moves to the destination file, and pastes it, but when the user repeats this exact same sequence of actions with the second subject line as the operand, Eager pops up and informs the user that it has detected a pattern. As the user performs the third copy action, Eager continuously marks the interface elements which it predicts that the user will operate on next, and the user can finally allow Eager to go ahead and complete the rest of the task, confident that it has induced the correct interaction pattern. Eager thus provides the user with information about what it will do before it goes and does it, and such "prospective feedback" is likely to be a necessary usability principle as long as this type of system does not have perfect inference capabilities.

4.12 Software Packaging

As discussed above, it is possible that future operating systems will become object-oriented and will abandon the model of monolithic applications as the way functionality is packaged. Because I am still using a system based on the traditional applications model, I currently have about six spelling checkers on my personal computer, since each application has its own. This profusion of spelling checkers leads to problems with inconsistent interfaces and the resulting increase in learning time and usage errors [89], and it requires me to update six different "personal" dictionaries with the specialized terms and proper names used in my writing. Also, my wealth of spelling checking functionality is restricted to work within some applications and does not help me when I am using others, such as my electronic mail package.

An object-oriented software structure would allow me to add various types of "language servers" to my system as needed, including a high-powered spelling checker, a thesaurus, and a grammar assistant. The increasing need to design user interfaces for international and multilingual use [93] certainly implies major benefits from an ability to exchange the language of the "language server" in a single location in the system and have the new language apply to all other system features without the need to reprogram them.

Readwear [51] is another example of the need for object-oriented packing of functionality. Readwear allows the computer to collect information about the time the user has spent working with various information objects at multiple levels of resolution, such as, for example, time spent writing the different chapters of a book, time spent editing assorted lines of code in a program, or time spent reading interesting netnews messages. Readwear recordings can then help users find information they previously worked on, it can help them keep in touch with other users working on the same project, and it can help them focus on the most important elements in a stream of news. For all of this to work, readwear needs to be recorded across applications and with the ability to associate data objects with each other on multiple levels, such as all the files constituting chapters in the book or the news items about the same event. It is thus apparent that readwear should be a service available through inheritance to all other parts of a system rather than being a feature in any specific application.

5 Impact on Usability Engineering Principles

Even though the next-generation user interfaces have a potential for increased usability, any given next-generation design might still have usability problems, in the same way as experience shows that the current generation of modern graphical interfaces can have usability problems [92]. Many well-known usability principles will probably continue to apply and and an initial analysis of a next-generation interface might be based on such known principles [131]. For example, a study of a handwriting system found that user performance increased significantly when better feedback was given [118]. Unfortunately, the next-generation interfaces that have been implemented so far have mostly not been subjected to user testing, so there is almost no data available with respect to the actual usability of these interfaces or regarding the new usability guidelines they may need.

In one of the few controlled studies comparing next-generation interfaces with current interfaces, Wolf [145] found that certain spreadsheet editing operations could be performed significantly faster with a pen-based gesture interface than with a traditional mouse-and-keyboard interface, and that the advantage of the pen was especially great for inexperienced users. Both experienced and inexperienced users had a mean task time of 13 seconds with the pen, experienced users had a mean task time of 18 seconds with the mouse and keyboard, and inexperienced users had a mean task time of 30 seconds with the mouse and keyboard. Other studies have found small improvements in user learning due to animated demonstrations and embedded help [4][105].

One should obviously beware of assuming that any advanced or fancy user interface technology automatically improves usability just because it can be referred to as "next-generation" according to certain criteria. As an example, I recently conducted a study of a paint program using sound effects to emphasize the result of its various features [98]. Younger users, including most of the research staff in our lab, were thrilled about the neat interface, but when we tested it with older users (70-75 years old), they found the interface more difficult to use when they were exposed to the sounds, possibly because they were overwhelmed by the multimedia effects.

Because of the lack of formal usability studies of the existing prototypes of next-generation interfaces, it is difficult to predict the usability engineering lifecycle [94][141] for these interfaces and to assess which usability methods will prove the most efficient in evaluating their usability. This section provides an initial discussion of these issues based on the limited available evidence and extrapolations from current usability engineering practice based on the characteristics of the new interfaces.

User testing of virtual reality interfaces could well present new difficulties. During testing of traditional, screen-based interfaces, the user and the experimenter have the screen as a shared reference for exchanging comments and questions: The experimenter can often directly observe which parts of the interface cause problems for the user, and the experimenter can point to parts of the screen and ask questions like, "what do you think this menu option will do?" [96].

In contrast, virtual reality interfaces envelop the user in its simulated world, excluding the experimenter who will often be relegated to watching a monitor with a 2-D replication of the image from one of the user's goggles. The experiences offered to the user and to the experimenter differ drastically, making it more difficult for the experimenter to hermeneutically empathize with the user and understand the user's problems. Even if the experimenter wore a second head-mounted display slaved to the user's display, the experience of rapidly being moved around a 3-D world (rather than moving yourself) could well be nauseating [113] and would certainly feel different from controlling your own movements.

These problems should certainly not prevent user testing of virtual reality interfaces, but they do indicate a need for special training for experimenters and possibly also for the invention of special tools to facilitate the test process. For example, a combination of an eye-tracker and a virtual reality system would allow the experimenter to observe what parts of the simulated world the user was currently watching. Also, a magic tele-pointer controlled by the experimenter might appear in the user's simulated space to point out objects when the experimenter wished to query the user's understanding of specific interface elements.

The difficulties in user testing some next-generation interfaces may mean greater reliance on the heuristic evaluation method [97] where user interface evaluators (preferably with some usability expertise) judge a design based on their own experience while using it and comparing it to a set of established usability heuristics. One such study where two interface specialists evaluated a virtual reality interface to a three dimensional brain model [79] indicated the presence of an interesting usability problem with respect to user navigation. Users wearing a headmounted display and a glove could move around an enlarged scanning of a human brain by two means. Smaller movements could be achieved by simply walking about the room, having the computer translate the user's physical movement to a movement in the brain model. The testers found "the correspondence between real-world movement and movement in the virtual world so accurate that there was no need for a conscious translation of intent to user interface action" [79], thus confirming the usability of the noncommand oriented aspect of the interface. The second navigation mechanism involved having the user point a finger as a gestural command to initiate "flying" through the brain. The testers found this very unnatural since they sometimes made the pointing gesture by mistake without wanting to fly, and since the feeling of the interface was not actually that of flying toward the brain but that of having the brain move toward them in the opposite direction of their pointing.

Most of the other next-generation user interface technologies are easier to test than virtual realities, so user testing should still form a major part of usability efforts for new interfaces. This is especially true given that we do not initially have good intuitions for what aspects of these interfaces will make for usable systems. It is likely that most of the traditional general principles for usable interface design [80][96] will continue to hold for next-generation interfaces, but new usability heuristics will probably also be needed. Some changes can certainly be expected. The traditional usability goal of consistency in the form of a single, uniform interface style [89] may be replaced with a goal of multiple interface styles where an appropriate style is chosen for each application, since noncommand interfaces may only be feasible with a tight match between the user interface and the user's task. Thus, external consistency [63] may become a more important factor than internal computer consistency, given that the user is supposed to focus on operating the domain and not on operating the computer.

Certainly, traditional user testing will be possible in many cases, even for virtual reality interfaces, if some care is shown in designing the experiment. For example, a virtual reality system for the planning of medical radiotherapy treatments is being subjected to rigorous user testing [17][18]. The application involves the treatment of tumors by hitting them with several beams of high-energy radiation that come from different directions, so that only the tumor gets the total dosage from all the beams, while healthy tissue is irradiated by only one beam. Since all patients and all tumors are different, treatments have to be planned on an individual basis as an inherently 3-D problem. A virtual reality interface allows the physician to position simulated beams through a semi-transparent 3-D model of the patient and to assess what beams pass through what organs. Since radiation oncologists have formal numerical measures of the quality of a given treatment plan, it is possible to statistically compare the result of using the virtual reality system with the result of using a control system that is currently in use. Even in domains where a scoring system is not conveniently available, it should be possible to conduct similar experiments where the same tasks are performed on next-generation and traditional systems, and the outcome compared at least informally.

Since next-generation interfaces are likely to be fairly complicated to implement (at least initially), it will be preferable to conduct usability studies at an early stage in the development lifecycle, such that as little development effort as possible is wasted on unusable designs. Such early testing is of course recommended also for traditional interfaces, but the exact methods to be used are likely to change somewhat. For example, there might be less reliance on paper mockups [91] due to the inherent dynamic and multimedia nature of some next-generation interfaces. At the same time, it is likely that early testing of next-generation designs will see increased use of Wizard of Oz [62] techniques where the intelligence of a human test assistant is used to simulate not-yet-implemented computer intelligence. A very elaborate Wizard of Oz-type experiment was performed at Carnegie Mellon University in 1990 to test a user's interaction with a computer-generated immersive interactive fiction [74]. As no such system can currently provide sufficient dramatic or interactive graphics quality, the entire interaction with the test user was simulated by three live human actors connected to a human director through wireless headsets.

Since many aspects of next-generation user interfaces are supposedly tightly related to the physical world, some forms of user testing may be conducted with physical objects instead of computers. For example, an early study of gestural interfaces for both pen-based interfaces and 3-D interfaces used a puzzle, a set of building blocks, and a doll with removable parts for the 3-D tasks instead of a virtual reality system [43]. The study found that users naturally generate a large number of different possible gestures for common editing operations, thus replicating the "verbal disagreement" finding from textual user interfaces, where a large number of different words are used by different people to describe the same command [39]. Similar studies comparing several applications found that certain user gestures applied to common functions across applications [8], thus showing that it is still possible to achieve consistency in gestural interfaces.

In addition to any fundamental usability problems or issues with noncommand interfaces, there will be initial transition problems as users have to transform their traditional computer skills to a new interaction paradigm. For example, a fairly new type of water faucets (found in many airport restrooms) turns on automatically when it senses the presence of a user's hands under the spout. This is a noncommand interface to the extent that water is supplied at the appropriate point of the hand-washing task without the need to explicitly ask for it. However, as pointed out by one usability analysis of water faucets [103], these new faucets may present problems to users who are accustomed to having to operate a handle to turn on the faucet. Such users will spend a long time searching for a knob or lever and may never try just waving their hands under the faucet. As a more computer-oriented example, I observed an experienced computer user trying to use a pen computer with a very nice interface design. This user had just installed a new software package on the computer and wanted to test it out. He proceeded to search all over the interface for the location of the application so that he could start it up. Only after failing to find the application did he realize that the way to access it was to use the "Create" menu to initiate a new document that was specified to include the functionality of the new application. Basically, the user was confused because the system used the object-oriented document model discussed earlier in this article and did not have the explicit representation of the concept of an "application" which he had been conditioned to expect based on his prior computer usage.

A similar problem was seen in the transfer from the previous generation of character-based interfaces to the current graphical interfaces. When graphical interfaces were first starting to see wide-spread use in the mid-1980s, I observed several experienced computer users being unable to rename files. They were used to having a separate operating system command for this operation and searched through all the system menus for a "rename" command. These users did not realize that the new principle of generic commands [114] made the change of a file name just one more instance of text editing to be accomplished by selecting the old name with the mouse and typing in the change.

In an analogy to these examples, one might expect initial users of noncommand interfaces to search for commands to direct secondary tasks that the computer would perform without any further instructions as soon as the users initiated their main use of the system. Also, some users might find it discomforting at first to have the computer second-guess their intentions.

Traditionally, the user has had the ultimate responsibility for ensuring that the dialogue was progressing in the desired direction. Therefore, interface designers could rely on having human ingenuity correct any mismatches between the system and the user's task. Obviously, such mismatches are never desirable, but under the command-oriented paradigm, the user could often put together commands in new and unexpected ways to work around any poorly designed features. For noncommand interfaces, the computer takes on much of the responsibility to react correctly, and the design will need to rely on a much more detailed task analysis to increase the probability that the computer's interpretations of the situation are indeed appropriate.

6 Conclusions

This article has identified twelve dimensions (listed in Table 2) across which next-generation user interfaces may differ from current user interfaces. As can be seen from the years in which the literature references listed below were published, the interface ideas discussed in this article are not all new, many of them being more than five years old. Even so, only recently have they reached a stage where they seem to define a direction for the next generation of user interfaces in the form of a true interface paradigm. Many next-generation interface ideas can be seen as contributing to the development of a generation of noncommand-based user interfaces which will be significantly different from the user interfaces in common use today.

To some extent, this article has been trying to predict the future, which is a notoriously difficult thing to do. Doubtlessly, some of the predictions will fail to come true, either because some of the ideas described here turn out to be too hard to make practical, or because some trends not described in the article (omitted as too esoteric or unforeseen altogether) turn out to have major impact on next-generation user interfaces anyway. As an exercise in retrospective prediction, one can consider how one would have predicted the future of user interfaces based on the state of the art at the time of Doug Engelbart's famous 1968 demonstration of the NLS system [29]. At that time, most of the element of current standard WIMP systems had been invented, including the mouse, windows, interactive computer graphics, hypertext, icons, and menus (though not pop-up menus) [106]. Not all of these elements were present in the NLS interface, however. NLS was based on a timeshared multi-user system, and it would probably have been hard to predict the current generation of personal computers from the 1968 demo. Instead, one would probably have predicted extensive use of tight integration between documents using pervasive hypertext capabilities, but this potential was (at least temporarily) lost in the transfer from centralized computing to personal computing supported by an exploding shrink-wrap software industry. Engelbart's 1968 demo included additional features such as an ability to overlay the data with live video images of other users which probably seemed convincing at the time, but which have not shown up in graphical user interfaces so far, with the exception of a few recent research efforts [55].

Admittedly, the examples of next-generation interfaces in this article are mainly from somewhat unusual application areas such as interactive fiction, games, naval display maps, computer music, and the planning of radiation treatment. It should be noted that these applications are important in their own right and form the basis for major industries with huge annual revenues. It is true that current applications of computers have tended to center around other human activities, but that does not necessarily imply that future uses will do so too. For example, consider the use of the technology of printing. Initial uses of printing may mostly have involved religious and scholarly applications, but current print technology is probably used much more for the printing of daily news, advertising, and fiction.

Traditional computer applications may also benefit from some of the next-generation user interface principles discussed in this article. A major research question to be resolved is whether the next generation of user interfaces will follow a single interface style for all applications or whether extremely disparate interface styles will be necessary for different tasks and usage situations. The previous generations of character-based interfaces had widely diverging and inconsistent interfaces, partly because their interactive bandwidth was so low that tailored interaction techniques were needed for each interface. In contrast, current graphical interfaces are powerful enough that consistent interface elements could be combined to satisfy most interaction needs, thus insuring that most applications have similar looks and feels. Future interfaces may reverse this trend towards interface uniformity because their even more expressive interface languages allow close matches between the interface and the user's task without the penalty suffered by users of inconsistent character-based dialogues.

An interesting question is when to expect regular use of next-generation user interfaces outside the research laboratories. To a small extent, some next-generation characteristics are already to be found in some present-day personal computer operating systems and applications, though the overwhelming feeling of using these systems is still that of traditional WIMP interfaces. Going to the other extreme, it is very unlikely that we will see the ultimate next-generation interface combining all the characteristics discussed in this article in a single system within the next ten years, which is about as far as one can predict in the computer field with a minimum of credibility. The major impediment to arriving at such a fully integrated next-generation system is probably the lack of sufficiently high-level data interchange and system integration standards, and such standards can hardly be defined before one knows what to aim for. In spite of these problems, it is likely that the user interfaces that will be released in the coming years will include more and more next-generation facilities, so the change will probably be evolutionary rather than a revolution as was the case when graphical user interfaces took over from character-based interfaces.

Acknowledgments

The author would like to thank Ralph Hill and Jim Hollan for helpful comments on previous versions of the manuscript. The author also thanks Roger Dannenberg for help with computer music references and Kent Wittenburg for help with gesture references.

References

1. Akscyn, R., McCracken, D., and Yoder, E. KMS: A distributed hypermedia system for managing knowledge in organizations. Communications of the ACM 31, 7 (July 1988), 820-835.

2. Anderson, J.R., and Skwarecki, E. The automated tutoring of introductory computer programming. Communications of the ACM 29, 9 (September 1986), 842-849.

3. Aspray, W., and Beaver, D. Marketing the monster: Advertising computer technology. Annals of the History of Computing 8, 2 (April 1986), 127-143.

4. Baecker, R.M., Small, I., and Mander, R. Bringing icons to life. In Proceedings ACM CHI'91 Conference Human Factors in Computing Systems (New Orleans, LA, 28 April-2 May 1991), 1-6.

5. Banning, J. Beyond the application program: A different approach to integrated software. BYTE 9, 1 (January 1984), 251-262.

6. Bell, G. Ultracomputers: A teraflop before its time. Communications of the ACM 35, 8 (August 1992), 26-47.

7. Biocca, F. Virtual reality technology: A tutorial. Journal of Communication 42, 4 (Autumn 1992), 23-72.

8. Blatt, L.A., and Schell, A. Gesture set economics for text and spreadsheet editors. In Proceedings Human Factors Society 34th Annual Meeting (1990), 410-414.

9. Bolt, R.A. 'Put-That-There': Voice and gesture at the graphics interface. ACM SIGGRAPH Computer Graphics 14, 3 (1980), 262-270.

10. Bos, E. Some virtues and limitations of action inferring mechanisms. In Proceedings ACM UIST'92 User

Interface Software and Technology (Monterey, CA, 15-18 November 1992), 79-88.

11. Bury, K.F., Boyle, J.M., Evey, R.J., and Neal, A.S. Windowing versus scrolling on a visual display terminal. Human Factors 24, 4 (August 1982), 385-394.

12. Buxton, W., and Myers, B. A study in two-handed input. In Proceedings ACM CHI'86 Conference Human Factors in Computing Systems (Boston, MA, 13-17 April 1986), 321-326.

13. Buxton, W., Dannenberg, R., and Vercoe, B. The computer as accompanist. In Proceedings ACM CHI'86 Conference Human Factors in Computing Systems (Boston, MA, 13-17 April 1986), 41-43.

14. Card, S.K., Robertson, G.G., and Mackinlay, J.D. The information visualizer: An information workspace. In Proceedings ACM CHI'91 Conference Human Factors in Computing Systems (New Orleans, LA, 28 April-2 May 1991), 181-188.

15. Carr, R., and Shafer, D. The Power of PenPoint, Addison-Wesley, Reading, MA, 1991.

16. Carroll, J.M., and Rosson, M.B. Paradox of the active user. In Carroll, J.M. (Ed.), Interfacing Thought: Cognitive Aspects of Human-Computer Interaction, MIT Press, Cambridge, MA, 1987, 80-111.

17. Chung, J.C. Application of head-mounted display to radiotherapy treatment planning. In Proceedings ACM CHI'91 Conference Human Factors in Computing Systems (New Orleans, LA, 28 April-2 May 1991), 489.

18. Chung, J.C. A comparison of head-tracked and non-head-tracked steering modes in the targeting of radiotherapy treatment beams. In Proceedings ACM 1992 Symposium on Interactive 3D Graphics (Cambridge, MA, 30 March - 1 April 1992), pp. 193-196 & p. 232.

19. Cruz-Neira, C., Sandin, D., DeFanti, T.A., Kenyon, R.V., and Hart, J.C. The Cave: Audio visual experience automatic virtual environment. Communications of the ACM 35, 6 (June 1992), 64-72.

20. Cypher, A. Eager: Programming repetitive tasks by example. In Proceedings ACM CHI'91 Conference Human Factors in Computing Systems (New Orleans, LA, 28 April-2 May 1991), 33-39.

21. Dannenberg, R. Real-time scheduling and computer accompaniment. In Mathews, M.V., and Pierce, J.R. (Eds.), Current Directions in Computer Music Research, Cambridge, MA: The MIT Press, 1989, 225-262.

22. Davis, J.R., and Schmandt, C.M. The Back Seat Driver: Real time spoken driving instructions. In Proceedings IEEE Vehicle Navigation and Information Systems Conference (Toronto, Canada, September 1989), 146-150.

23. Deerwester, S., Dumais, S.T., Landauer, T.K., Furnas, G.W., and Harshman, R.A. Indexing by latent semantic analysis. Journal of the Society for Information Science 41, 6 (1990), 391-407.

24. Dourish, P., and Bly, S. Portholes: Supporting awareness in a distributed work group. In Proceedings ACM CHI'92 Conference Human Factors in Computing Systems (Monterey, CA, 3-7 May 1992), 541-547.

25. Dreger, L., Grauman, A., Ho, T., Howlett, V., Lehmann, R., Malamud, M., Marceau, R., and Tobey, C. An object-oriented evolution of Windows: Information at your fingertips (videotape). ACM SIGGRAPH Video Review 78 (1992).

26. Druin, A. Noobie: The animal design playstation. ACM SIGCHI Bulletin 20, 1 (July 1988), 45-53.

27. Dubberly, H., and Mitsch, D. Knowledge navigator (videotape). ACM SIGGRAPH Video Review 79 (published 1992, tape made in 1987).

28. Dubberly, H., and Mitsch, D. HyperCard 1992 (videotape). ACM SIGGRAPH Video Review 79 (published 1992, tape made in 1988).

29. Engelbart, D.C., and English, W.K. A research center for augmenting human intellect. In Proceedings of the Fall Joint Computer Conference, AFIPS Conference Proceedings 33 (San Francisco, CA, December 1968), 395-410.

30. Erickson, T., and Salomon, G. Designing a desktop information system: Observations and issues. In Proceedings ACM CHI'91 Conference Human Factors in Computing Systems (New Orleans, LA, 28 April-2 May 1991), 49-54.

31. Feiner, S., and Shamash, A. Hybrid user interfaces: breeding virtually bigger interfaces for physically smaller computers. In Proceedings of ACM UIST'91 Symposium on User Interface Software and Technology (Hilton Head, SC, 11-13 November 1991), 9-17.

32. Feiner, S., MacIntyre, B., and Seligmann, D. Knowledge-based augmented reality. Communications of the ACM 36, 7 (July 1993), 52-62.

33. Fischer, G., Lemke, A., and Schwab, T. Knowledge-based help systems. In Proceedings ACM CHI'85 Conference Human Factors in Computing Systems (San Francisco, CA, 14-18 April 1985), 161-167.

34. Fisher, S.S. Virtual interface environments. In Laurel, B. (Ed.), The Art of Human-Computer Interface Design, Addison-Wesley, Reading, MA, 1990, 423-438.

35. Furnas, G.W. Generalized fisheye views. In Proceedings ACM CHI'86 Conference Human Factors in Computing Systems (Boston, MA, 13-17 April 1986), 16-23.

36. Furnas, G.W. Graphical reasoning for graphical interfaces (videotape). ACM SIGGRAPH Video Review 56 (1990), clip 4.

37. Furnas, G.W. New graphical reasoning models for understanding graphical interfaces. In Proceedings ACM CHI'91 Conference on Human Factors in Computing Systems (New Orleans, LA, 30 April-2 May 1991), 71-78.

38. Furnas, G.W. Top level editing vs. top level information retrieval: Scale in information manipulation. In Digest of Position Papers, ACM CHI'92 Research Symposium (Monterey, CA, 1-2 May 1992).

39. Furnas, G.W., Landauer, T.K., Gomez, L.M., and Dumais, S.T. The vocabulary problem in human-system communication. Communications of the ACM 30, 11 (November 1987), 964-971.

40. Gates, B. Information at Your Fingertips (videotape). Microsoft, Redmond, WA, 1990.

41. Gaver, W.W. The SonicFinder: An interface that uses auditory icons. Human-Computer Interaction, 4, 1 (1989), 67-94.

42. Gaver, W.W., and Smith, R.B. Auditory icons in large-scale collaborative environments. In Proceedings INTERACT'90 3rd IFIP Conference Human-Computer Interaction (Cambridge, U.K., 27-31 August 1990), 735-740.

43. Gould, J.D., and Salaun, J. Behavioral experiments on handmarkings. ACM Transactions on Office Information Systems 5, 4 (October 1987), 358-377.

44. Green, M., and Jacob, R. SIGGRAPH'90 workshop report: Software architectures and metaphors for non-WIMP user interfaces. ACM SIGGRAPH Computer Graphics 25, 3 (July 1991), 229-235.

45. Greenberg, S., and Whitten, I.H. Adaptive personalized interfaces -- A question of viability. Behaviour & Information Technology 4, 1 (January 1985), 31-45.

46. Grudin, J. Why CSCW applications fail: Problems in the design and evaluation of organizational interfaces. In Proceedings of ACM CSCW'88 Conference on Computer-Supported Cooperative Work (Portland, OR, 26-28 September 1988), 85-93.

47. Grudin, J. The computer reaches out: The historical continuity of interface design. In Proceedings ACM CHI'90 Conference Human Factors in Computing Systems (Seattle, WA, 1-5 April 1990), 261-268.

48. Helm, R., Marriott, K., and Odersky, M. Building visual language parsers. In Proceedings ACM CHI'91 Conference on Human Factors in Computing Systems (New Orleans, LA, 30 April-2 May 1991), 105-112.

49. Henderson, D.A., and Card, S.K. Rooms: The use of multiple virtual workspaces to reduce space

contention in a window-based graphical user interface. ACM Transactions on Graphics 5, 3 (1986), 211-243.

50. Hill, R.D. Event-response systems: A technique for specifying multi-threaded dialogues. In Proceedings ACM CHI+GI'87 Conference Human Factors in Computing Systems and Graphics Interface (Toronto, Canada, 5-9 April 1987), 241-248.

51. Hill, W.C., and Hollan, J.D. Edit wear and read wear. In Proceedings ACM CHI'92 Conference Human Factors in Computing Systems (Monterey, CA, 3-7 May 1992), 3-9.

52. Hirata, K., Aoyagi, T., and Konaka, H. How to realize jazz feelings: A logic programming approach. In Proceedings International Conference Fifth Generation Computer Systems (Tokyo, Japan, 28 November-2 December 1988), 1081-1088.

53. Hollnagel, E. The design of fault tolerant systems: Prevention is better than cure. In Proceedings Second European Meeting on Cognitive Science Approaches to Process Control (Siena, Italy, 24-27 October 1989).

54. Hutchinson, T. E., and Jacob, R. J. K. Eye-gaze computer interfaces. IEEE Computer 26, 7 (July 1993), 65-67.

55. Ishii, H., and Kobayashi, M. ClearBoard: A seamless medium for shared drawing and conversation with eye contact. In Proceedings ACM CHI'92 Conference Human Factors in Computing Systems (Monterey, CA, 3-7 May 1992), 525-532.

56. Jacob, R.J.K. What you look at is what you get: Eye movement-based interaction techniques. In Proceedings ACM CHI'90 Conference Human Factors in Computing Systems (Seattle, WA, 1-5 April 1990), 11-18.

57. Jacob, R.J.K. The use of eye movements in human-computer interaction techniques: What you look at is what you get. ACM Trans. Information Systems 9, 2 (April 1991), 152-169.

58. Johnson, J.A., Nardi, B.A., Zarmer, C.L., and Miller, J.R. ACE: Building interactive graphical applications. Communications of the ACM 36, 4 (April 1993), 41-55.

59. Jørgensen, A.H., and Sauer, A. The personal touch: A study of users' customization practice. In Proceedings INTERACT'90 3rd IFIP Conference Human-Computer Interaction (Cambridge, U.K., 27-31 August 1990), 549-554.

60. Kay, A. User interface: A personal view. In Laurel, B. (Ed.), The Art of Human-Computer Interface Design, Addison-Wesley, Reading, MA, 1990, 191-207.

61. Kay, A., and Goldberg, A. Personal dynamic media. IEEE Computer 10, 3 (March 1977), 31-41.

62. Kelley, J.F. An iterative design methodology for user-friendly natural language office information applications. ACM Trans. Office Information Systems 2, 1 (January 1984), 26-41.

63. Kellogg, W.A. The dimensions of consistency. In Nielsen, J. (Ed.), Coordinating User Interfaces for Consistency, Academic Press, Boston, MA, 1989, 9-20.

64. Kellogg, W.A., Carroll, J.M., and Richards, J.T. Making reality a cyberspace. In Benedikt, M. (Ed.), Cyberspace: First Steps, MIT Press, Cambridge, MA, 1991, 411-431.

65. Krueger, M. VIDEOPLACE -- An artificial reality. In Proceedings ACM CHI'85 Conference Human Factors in Computing Systems (San Francisco, CA, 14-18 April 1985), 35-40.

66. Kurlander, D., and Feiner, S. Editable graphical histories: The video. ACM SIGGRAPH Video Review 63, 1991.

67. Kurlander, D., and Feiner, S. A history-based macro by example system. In Proceedings ACM UIST'92 User Interface Software and Technology (Monterey, CA, 15-18 November 1992), 99-106.

68. Kurtenbach, G., and Hulteen, E.A. Gestures in human-computer communication. In Laurel, B. (Ed.), The Art of Human-Computer Interface Design, Addison-Wesley, Reading, MA, 1990, 309-317.

69. Lai, K-Y., Malone, T.W., and Yu, K-C. Object Lens: A "spreadsheet" for cooperative work. ACM Transactions on Office Information Systems, 6, 4 (October 1988), 332-353.

70. Landauer, T.K., Egan, D.E., Remde, J.R., Lesk, M.E., Lochbaum, C.C., and Ketchum, R.D. Enhancing the usability of text through computer delivery and formative evaluation: The SuperBook project. In McKnight, C., Dillon, A., and Richardson, J. (Eds.), Hypertext: A Psychological Perspective, Ellis Horwood, 1993.

71. Lansdale, M.W. The psychology of personal information management. Applied Ergonomics 19, 1 (1988), 55-66.

72. Lansdale, M.W., and Edmonds, E. Using memory for events in the design of personal filing systems. International Journal of Man-Machine Studies 36, 1 (January 1992), 97-126.

73. Laurel, B. Interface agents: Metaphors with character. In Laurel, B. (Ed.), The Art of Human-Computer Interface Design, Addison-Wesley, Reading, MA, 1990, 355-365.

74. Laurel, B. Computers as Theatre. Addison-Wesley, Reading, MA, 1991, pp. 189-191.

75. Laurel, B., Oren, T., and Don, A. Issues in multimedia interface design: Media integration and interface agents. In Proceedings ACM CHI'90 Conference Human Factors in Computing Systems (Seattle, WA, 1-5 April 1990), 133-139.

76. Lee, A., and Lochovsky, F.H. Enhancing the usability of an office information system through direct manipulation. In Proceedings ACM CHI'83 Conference Human Factors in Computing Systems (Boston, MA, 12-15 December 1983), 130-134.

77. MacLean, A., Carter, K., Lšvstrand, L., and Moran, T. User-tailorable systems: Pressing the issues with buttons. In Proceedings ACM CHI'90 Conference Human Factors in Computing Systems (Seattle, WA, 1-5 April 1990), 175-182.

78. Mander, R., Salomon, G., and Wong, Y.Y. A 'pile' metaphor for supporting casual organization of information. In Proceedings ACM CHI'92 Conference Human Factors in Computing Systems (Monterey, CA, 3-7 May 1992), 627-634.

79. Mercurio, P.J., and Erickson, T.D. Interactive scientific visualization: An assessment of a virtual reality system. In Proceedings INTERACT'90 3rd IFIP Conference Human-Computer Interaction (Cambridge, U.K., 27-31 August 1990), 741-745.

80. Molich, R., and Nielsen, J. Improving a human-computer dialogue. Communications of the ACM 33, 3 (March 1990), 338-348.

81. Morita, H., Hashimoto, S., and Ohteru, S. A computer music system that follows a human conductor. IEEE Computer 24, 7 (July 1991), 44-53.

82. Muller, M.J. Multifunctional cursor for direct manipulation user interfaces. In Proceedings ACM CHI'88 Conference Human Factors in Computing Systems (Washington, DC, 15-19 May 1988), 89-94.

83. Myers, B.A. Creating User Interfaces by Demonstration. Academic Press, Boston, MA, 1988.

84. Myers, B.A. Taxonomies of visual programming and program visualization. Journal of Visual Languages and Computing 1, 1 (March 1990), 97-123.

85. Myers, B.A. Demonstrational interfaces: A step beyond direct manipulation. IEEE Computer 25, 8 (August 1992), 61-73.

86. Myers, B.A., Cypher, A., Maulsby, D., Smith, D.C., and Shneiderman, B. Demonstrational interfaces: Coming soon? In Proceedings ACM CHI'91 Conference Human Factors in Computing Systems (New Orleans, LA, 28 April-2 May 1991), 393-396.

87. Negroponte, N. An iconoclastic view beyond the desktop metaphor. International Journal of Human-Computer Interaction 1, 1 (1989), 109-113.

88. Newman, W., and Wellner, P. A desk supporting computer-based interaction with paper documents. In Proceedings ACM CHI'92 (Monterey, CA, 3-7 May 1992), 587-592.

89. Nielsen, J. (Ed.). Coordinating User Interfaces for Consistency. Academic Press, Boston, MA, 1989.

90. Nielsen, J. Multimedia and Hypertext. Academic Press, Boston, MA, 1995.

91. Nielsen, J. Paper versus computer implementations as mockup scenarios for heuristic evaluation. In Proceedings INTERACT'90 3rd IFIP Conference Human-Computer Interaction (Cambridge, U.K., 27-31 August 1990), 315-320.

92. Nielsen, J. Traditional dialogue design applied to modern user interfaces. Communications of the ACM 33, 10 (October 1990), 109-118.

93. Nielsen, J. (Ed.). Designing User Interfaces for International Use. Elsevier Science Publishers, Amsterdam, the Netherlands, 1990.

94. Nielsen, J. The usability engineering life cycle. IEEE Computer 25, 3 (March 1992), 12-22.

95. Nielsen, J. Noncommand user interfaces. Communications of the ACM 36, 4 (April 1993), 83-99.

96. Nielsen, J. Usability Engineering. Academic Press, Boston, MA, 1993.

97. Nielsen, J., and Molich, R. Heuristic evaluation of user interfaces. In Proceedings ACM CHI'90 Conference Human Factors in Computing Systems (Seattle, WA, 1-5 April 1990), 249-256.

98. Nielsen, J., and Schaefer, L. Sound effects as an interface element for older users. Behaviour & Information Technology 12, 4 (July-August 1993), 208-215.

99. Nielsen, J., Mack, R.L., Bergendorff, K.H., and Grischkowsky, N.L. Integrated software in the professional work environment: Evidence from questionnaires and interviews. In Proceedings ACM CHI'86 Conference Human Factors in Computing Systems (Boston, 13-17 April 1986), 162-167.

100. Nielsen, J., Bush, R.M., Dayton, T., Mond, N.E., Muller, M.J., and Root, R.W. Teaching experienced developers to design graphical user interfaces. In Proceedings ACM CHI'92 Conference Human Factors in Computing Systems (Monterey, CA, 3-7 May 1992), 557-564.

101. Nilsen, E. Perceptual-motor control in human-computer interaction. ACM SIGCHI Bulletin 23, 4 (October 1991), 83-84.

102. Nonogaki, H., and Ueda, H. FRIEND21 project: A construction of 21st century human interface. In Proceedings ACM CHI'91 Conference Human Factors in Computing Systems (New Orleans, LA, 28 April-2 May 1991), 407-414.

103. Norman, D.A. The Design of Everyday Things. Basic Books, New York, 1988, pp. 166-172.

104. Pausch, R. Virtual reality on five dollars a day. In Proceedings ACM CHI'91 Conference Human Factors in Computing Systems (New Orleans, LA, 28 April-2 May 1991), 265-270.

105. Payne, S.J., Chesworth, L., and Hill, E. Animated demonstrations for exploratory learners. Interacting with Computers 4, 1 (April 1992), 3-22.

106. Perry, T.S., and Voelcker, J. Of mice and menus: Designing the user-friendly interface. IEEE Spectrum 26, 9 (September 1989), 46-51.

107. Press, L. Dynabook revisited -- portable computers past, present and future. Communications of the ACM 35, 3 (March 1992), 25-32.

108. Rao, R., Card, S., Jellinek, H., Mackinlay, J., and Robertson, G. The information grid: A retrieval-top-level extension to the desktop user interface metaphor. In Proceedings ACM UIST'92 User Interface Software and Technology (Monterey, CA, 15-18 November 1992), 23-32.

109. Rheingold, H. Virtual Reality. Summit Books, NY, 1991.

110. Rhyne, J.R., and Wolf, C.G. Recognition-based user interfaces. In Hartson, H.R., and Hix, D. (Eds.), Advances in Human-Computer Interaction Vol. 4, Ablex, Norwood, NJ, 1993, 191-250.

111. Roads, C. Research in music and artificial intelligence. ACM Computing Surveys 17, 2 (June 1985), 163-190.

112. Rohall, S.L., Patterson, J.F., and Hill, R.D. Go fish! A multi-user game in the Rendezvous system. ACM SIGGRAPH Video Review 76, 1992.

113. Rolnick, A., and Lubow, R.E. Why is the driver rarely motion sick? The role of controllability in motion sickness. Ergonomics 34, 7 (1991), 867-879.

114. Rosenberg, J.K., and Moran, T.P. Generic commands. In Proceedings of INTERACT'84 First IFIP Conference on Human-Computer Interaction (London, U.K., 4-7 September 1984), 245-249.

115. Rubine, D. Combining gestures and direct manipulation. In Proceedings ACM CHI'92 Conference Human Factors in Computing Systems (Monterey, CA, 3-7 May 1992), 659-660.

116. Salomon, G., Oren, T., and Kreitman, K. Using guides to explore multimedia databases. In Proceedings of 22nd Hawaii International Conference on System Sciences (Kailua-Kona, HI, 3-6 January 1989), 3-12.

117. Sandewall, E. Programming in an interactive environment: The "Lisp" experience. Computing Surveys 10, 1 (March 1978), 35-71.

118. Santos, P.J., Baltzer, A.J., Badre, A.N., Henneman, R.L., and Miller, M.S. On handwriting recognition system performance: Some experimental results. Proc. Human Factors Society 36th Annual Meeting (Atlanta, GA, 12-16 October 1992), 283-287.

119. Schmandt, C. Conversational desktop (videotape). ACM SIGGRAPH Video Review 27 (1987).

120. Schmandt, C., and Arons, B. A conversational telephone messaging system. IEEE Transactions on Consumer Electronics, CE-30, 3 (1984), xxi-xxiv.

121. Schmandt, C.M., and Davis, J.R. Synthetic speech for real time direction-giving. IEEE Transactions on Consumer Electronics 35, 3 (August 1989), 649-653.

122. Sellen, A., and Nicol, A. Building user-centered online help. In Laurel, B. (Ed.), The Art of Human-Computer Interface Design, Addison-Wesley, Reading, MA, 1990, 143-
153.

123. Sellen, A.J., Kurtenbach, G.P., and Buxton, W.A.S. The prevention of mode errors through sensory feedback. Human-Computer Interaction 7, 2, 141-164.

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

125. Starker, I., and Bolt, R.A. A gaze-responsive self-disclosing display. In Proceedings ACM CHI'90 Conference Human Factors in Computing Systems (Seattle, WA, 1-5 April
1990), 3-9.

126. Steuer, J. Defining virtual reality: Dimensions determining telepresence. Journal of Communication 42, 4 (Autumn 1992), 73-93.

127. Sukaviriya, P.N., and Foley, J.D. Coupling a UI framework with automatic generation of context-sensitive animated help. In Proceedings of ACM UIST'90 Third Annual Symposium on User Interface Software and Technology (Snowbird, UT, 3-5 October 1990), 152-166.

128. Tanner, P.P., MacKay, S.A., Steward, D.A., and Wein, M. A multitasking switchboard approach to user interface management. In Proceedings ACM SIGGRAPH'86 (Dallas, TX, Aug. 1986), 241-248.

129. Teitelman, W. Do what I mean: The programmer's assistant. Computers and Automation 21 (April

1972), 8-11.

130. Tesler, L.G. Networked computing in the 1990s. Scientific American 265, 3 (September 1991), 86-93.

131. Thomas, J.C., and Stuart, R. Virtual reality and human factors. Proc. Human Factors Society 36th Annual Meeting (Atlanta, GA, 12-16 October 1992), 207-210.

132. Tucker, P., and Jones, D.M. Voice as interface: An overview. International Journal of Human-Computer Interaction 3, 2 (1991), 145-170.

133. Vercoe, B., and Puckette, M. Synthetic rehearsal: Training the Synthetic Performer. In Proceedings of the 1985 International Computer Music Conference (Vancouver,
Canada, August 1985), Computer Music Association, 275-278.

134. Vincent, V.J., and Jacobsen, L. The Mandala system: An alternative approach to virtual reality. In Jacobsen, L. (Ed.), Cyberarts: Exploring Art & Technology, Miller Freeman Inc., San Francisco, CA, 264-271.

135. Want, R., Hopper, A., Falcão, V., and Gibbons, J. The active badge location system. ACM Transactions on Information Systems 10, 1 (January 1992), 91-102.

136. Watnabe, T., and Yuuki, N. A voice reaction system with a visualized response equivalent to nodding. In Smith, M.J., and Salvendy, G. (Eds.), Work with Computers: Organizational, Management, Stress, and Health Aspects, Amsterdam, the Netherlands: Elsevier Science Publishers, 1989, 396-403.

137. Weiser, M. The computer for the 21st century. Scientific American 265, 3 (September 1991), 94-104.

138. Weiser, M. Some computer science issues in ubiquitous computing. Communications of the ACM 36, 7 (July 1993), 74-84.

139. Wellner, P. The DigitalDesk calculator: Tangible manipulation on a desk top display. In Proceedings ACM UIST'91 Symposium on User Interface Software and Technology (Hilton Head, SC, 11-13 April 1991), 27-33.

140. Wellner, P. Interacting with paper on the DigitalDesk. Communications of the ACM 36, 7 (July 1993), 86-97.

141. Whiteside, J. Bennett, J., and Holtzblatt, K. Usability engineering: Our experience and evolution. In Helander, M. (Ed.), Handbook of Human-Computer Interaction, Elsevier Science Publishers, Amsterdam, the Netherlands, 1988, pp. 791-817.

142. Williamson, C., and Shneiderman, B. The Dynamic HomeFinder: Evaluating dynamic queries in a real-estate information exploration system. In Proceedings ACM SIGIR'92 Fifteenth International Conference on Research and Development in Information Retrieval (Copenhagen, Denmark, 21-24 June 1992), 338-346.

143. Wittenburg, K., Weitzman, L., and Talley, J. Unification-based grammars and tabular parsing for graphical languages. Journal of Visual Languages and Computing 2 (1991), 347-370.

144. Wolf, C.G. The paper-like interface (videotape). ACM SIGGRAPH Video Review 47 (1989), clip 1.

145. Wolf, C.G. A comparative study of gestural, keyboard, and mouse interfaces. Behaviour & Information Technology 11, 1 (January 1992), 13-23.

146. Wolf, C.G., Rhyne, J.R., and Ellozy, H.A. The paper-like interface. In Salvendy, G. and Smith, M.J. (Eds.): Designing and Using Human-Computer Interfaces and Knowledge Based Systems, Amsterdam, the Netherlands: Elsevier Science Publishers, 1989, 494-501.

Topics: Human Computer Interaction Predictions & Milestones Technology

Learn More

Training Courses

UX Basic Training

- Interaction Design: 3-Day Course
- User Interface Principles Every Designer Must Know

Articles

- Mental Models
- 10 Usability Heuristics for User Interface Design
- The Human Body as Touchscreen Replacement
- When the UI Is Too Fast
- Mobile Usability for Cats: Essential Design Principles for Felines