Open Syntax: Improving Access for All Users

Robert J.K. Jacob

Department of Electrical Engineering and Computer Science
Tufts University
Medford, Mass. USA

and

Tangible Media Group
MIT Media Laboratory
Cambridge, Mass. USA

ABSTRACT
Trends in new multi-modal user interfaces and pervasive mobile computing are raising technical problems for building flexible interfaces that can adapt to different communication modes. I hope to show how some aspects of the technical solutions that will be needed for these problems will also help to solve problems of access for elderly users.

General Terms
Human Factors, Standardization, Languages.

Keywords
User interface management system, syntax, semantics, universal access, dialogue independence, multi-modal interaction.

1. INTRODUCTION
A basic problem for both pervasive mobile computing and for universal access is to provide a wide range of different access and interaction mechanisms that reach the same underlying data and functionality. It applies when providing new more powerful multi-modal interfaces to common applications for improved interaction. Here the goal is to improve the interface to an application. It also applies to providing alternative cross-modal access to applications for mobile users. Here the need is to accommodate users with temporarily limited interaction facilities as they drive or walk or use a telephone by adapting the interface to their context. And it also applies to the problem of providing access to elderly users who might have physical, cognitive, or other limitations or differences compared to the users for whom the original interfaces were designed. All three of these cases require an interchangeable or adaptable user interface front end, accessing a standard application back end. All three of these problems can share a similar technical solution for designing software structures for building user interfaces.

WUAUC'01, May 22-25, 2001, Alcácer do Sal, Portugal.

This paper addresses lower-level interaction issues, where we seek to provide customized access to common facilities. We want to provide different, specialized forms of "nuts and bolts" interface to the same underlying data or functions—so that elderly users can share information and functions with the rest of the community. There will also be cases where more thoroughly specialized functions or facilities might be provided just for elderly users, specialized not just at the lower levels, but in their conceptual models as well, but we do not address this part of the landscape here.

2. SYNTAX AND SEMANTICS
The first step is to decompose user interface design into the semantic, syntactic, and lexical levels, as first described by Foley and Wallace[3, 4]:

The *semantic level* describes the functions performed by the system. This corresponds to a description of the functional requirements of the system. It specifies what information is needed for each operation on an object, what errors can occur, how the errors are handled, and what the results of each operation are—but it does not address how the user will invoke the functions. The semantic level defines "meanings," rather than "forms" or "sequences," which are left to the lower levels. It provides the high-level model or abstraction of the functions of the system.

The *syntactic level* describes the sequences of inputs and outputs necessary to invoke the functions described. That is, it gives the rules by which which sequences of words (indivisible units of meaning or "tokens") in the language are formed into proper (but not necessarily semantically meaningful) sentences. The design of the syntactic level describes the sequence of the logical input, output, and semantic operations, but not their internal details. A logical input or output operation is an input or output token. The tokens cannot be further decomposed without loss of meaning. For example, the mouse movements and mouse button clicks needed to make a menu selection do not individually provide information to the application. Its internal structure is described at the lexical level, while the syntactic describes when the user may enter it and what will happen next if he or she does (for an input token) or when the system will produce it (for an output token).

The *lexical level* determines how the inputs and outputs are actually formed from primitive hardware operations or lexemes. It represents the binding of hardware actions to the hardware-

independent tokens of the input and output languages. While tokens are the smallest units of meaning with respect to the syntax of the dialogue, lexemes are the actual hardware input and output operations that comprise the tokens.

As a simple illustration, consider an automated teller machine (ATM). The semantic level defines the basic commands the user can execute, such as logging in, withdrawing cash, and making deposits. It specifies each command and its inputs and outputs, but carefully avoids specifying *how* the command is to be invoked. The syntactic level defines the permissible sequences of input tokens to invoke each command. For example, a withdrawal command might require a token for the command itself, one for which account to use, and another for the amount. The syntax prescribes all the acceptable orderings for these tokens. Finally, the lexical level defines each of the tokens in terms of a primitive hardware action. For example, the withdraw command itself might be a dedicated function key, the account choice might be made with a touch screen, and the amount might be entered with a keypad.

Web browsing provides another example of these distinctions. The semantic level of the design is embodied in the content presented in the web page, the command for requesting another page, and the command to transmit a filled-in data form page. The syntactic and lexical levels are concerned with the user interface for examining the web page—scrolling, rearranging, changing colors and styles locally, and even navigating through a downloaded 3D world—as well as maintaining personal history lists, hotlists, or bookmarks, and manipulating data items on a downloaded form. A different user interface (browser) can be substituted, changing the syntactic and lexical layers, without affecting the semantic layer (web servers).

Shneiderman's *syntactic-semantic object-action model* is a related approach, which also separates the task and computer concepts (i.e., the semantics) from the syntax for carrying out the task[17]. For example, the task of writing a scientific journal article can be decomposed into the sub-tasks for writing the title page, the body, and the references. Similarly, the title page might be decomposed into a unique title, one or more authors, an abstract, and several keywords. To write a scientific article, the user must understand these task semantics. To use a word processor, the user must learn about computer semantics, such as directories, filenames, files, and the structure of a file. Finally, the user must learn the syntax of the commands for opening a file, inserting text, editing, and saving or printing the file. Novices often struggle to learn how to carry out their tasks on the computer and to remember the syntactic details. Once learned, the task and computer semantics are relatively stable in human memory, but the syntactic details must be frequently rehearsed. A knowledgeable user of one word processor who wishes to learn a second one only needs to learn the new syntactic details. More importantly, the same should apply to a user who wants to access the same word processing functionality from a PDA or non-visually while driving a car or using a large-print interface or perhaps one with a menu structure designed to reduce short-term memory load.

3. USER INTERFACE SOFTWARE

A key principle in user interface software is *dialogue independence*, the notion of separating all user interface-related code from the rest of the application code, first described by Hartson and Hix[6]. This allows changes to be made to the dialogue design without affecting the application code, and thus makes easier the repeated changes to prototype interfaces that are often needed. Using our multi-level model, this means that the code for the semantic level is separated from the code for the syntactic and lexical levels through a well-defined software interface (see Figure 1).

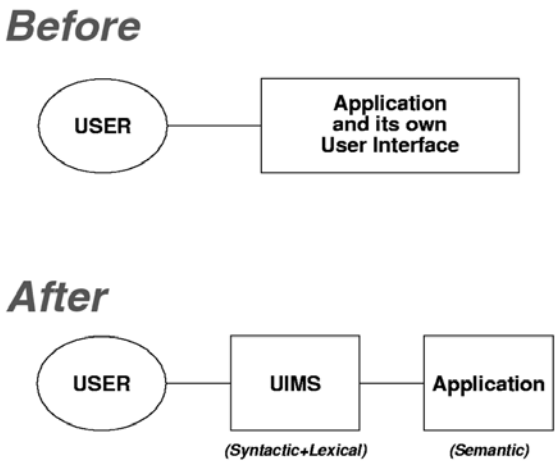

Figure 1. Dialogue independence, using a user interface management system.

A user interface management system (UIMS) is a software component that is separate from the application program that performs the underlying task[14]. The UIMS conducts the interaction with the user, implementing the syntactic and lexical levels, while the rest of the system implements the semantic level. Like an operating system or graphics library, a UIMS separates functions used by many applications and moves them to a shared subsystem. It centralizes implementation of the user interface and permits some of the effort of designing tools for user interfaces to be amortized over many applications and shared by them. It also encourages consistent "look and feel" in user interfaces to different systems, since they share the user interface component. A UIMS also supports the concept of dialogue independence, where changes can be made to the interface design (the user-computer dialogue) without affecting the application code. This supports the development of alternative user interfaces for the same application (semantics), which facilitates both iterative refinement of the interface through prototyping and testing and, in the future, alternative interfaces for users of different physical or other disabilities (see Figure 2).

The key ingredient of UIMSs and other high-level user interface software tools is a way of describing and implementing the interactive behavior or dialogue sequencing. The choice of specification language or model used for this is thus the key to UIMS design. Permissible sequences of user actions can be defined in a variety of ways. Whether such methods are literally linguistic or interactive, they generally fall under the rubric of

user interface description languages (UIDLs). Most of these are based on various kinds of special-purpose languages for interface design[9].

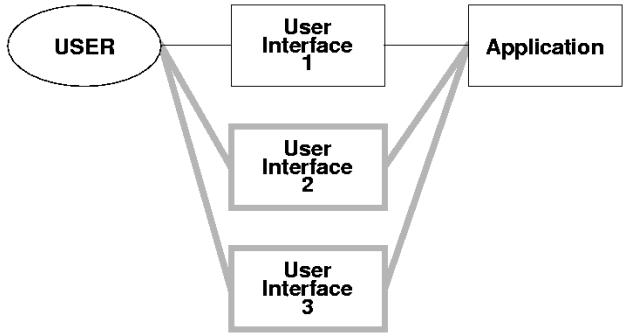

Figure 2. Alternate interfaces to same application.

With a UIMS, the code for an interactive application consists of two parts: a dialogue component, which handles communication with the user and implements the syntactic and lexical levels of the system; and a computational component, which performs the functional processing and implements the semantic level[10]. The semantic level of an application developed with a UIMS is typically written as a set of subroutines. The UIMS is responsible for calling appropriate routines in response to user inputs. In turn, the routines influence the dialogue—for instance, by modifying what the user can do next on the basis of the outcome of a computation. External control is a simpler, less powerful alternative, where the action routines have no influence over the flow of the dialogue; control resides solely in the UIMS. Component software technologies such as COM, CORBA, and JavaBeans, provide platforms for implementing shared control; the UIMS and application can communicate in both directions through the software component interface.

4. OPEN SYNTAX

Given this framework for thinking about user interface design and software, a solution becomes clear. We need to provide alternative front ends to a common back end for a given application. That is, interchangeable syntactic/lexical modules, communicating with a single semantic module. This in turn requires that interactive programs be opened up, so that their front and back ends can be separated. While many modern GUI programs are built using toolkits and user interface management systems that facilitate this split, they rarely provide runtime (or any) access to the internal dialogue between their front and back ends. Open Syntax seeks to provide runtime access to the software interface between front and back end, syntax and semantics, interface and application,

The ultimate goal is to provide a retargetable syntax module that can adapt to the user in several ways: It might adapt to different contexts or situations as the user moves around, such as office, car, airplane, shopping mall (using a PDA), or war room. When the system is started, it would dynamically determine the user's current situation and the input and output devices and channels available for interaction, and provide an appropriately tailored user interface. The interface could similarly adapt to different user abilities, to accommodate temporary and permanent physical and cognitive characteristics of elderly users. For example, it might provide a large-type interface. More interestingly, it might provide an interface with an alternate syntax designed to reduce short-term memory load. In each case, the change is made only to the front end or syntax of the interface; the back end or semantics remains the same.

To accomplish this, we need to separate the syntactic and semantic portions of an interactive application. We have described the technical means and intellectual framework needed to separate the two. Unfortunately, too many applications intertwine them, because it is often expedient and may provide prettier looking interfaces as well as proprietary benefits to software companies. What is required is to allow access past the syntax directly to core functionality or semantics in an application-independent way.

For example, it is usually difficult to access the functions of a word processing program without going through its specific user interface. Some success has been achieved with email programs, which use standard protocols (POP, IMAP) for accessing message retrieval and filing functions. Any IMAP email client can access any IMAP server, with generally good interoperability. Moreover, email in plain text and MIME is even more application-independent. Any email program can generally send mail to be read by any other email program, with a well-accepted set of standards for the "semantic" functions inherent in all email. Nevertheless, software vendors continue to try to add non-standard "enhancements" to their email, which can only be accessed through the user interface of their own product. This has obvious business advantages, but defeats the broader goal of adaptable access to basic functions. Calendar programs have also begun to provide generic functions, so that they can share data between office computers, PDAs, and web calendar servers. Spreadsheets have not reached this stage.

An analogy can be found in text markup—the distinction between markup based on appearance (bold, italic) and that based on content (title, heading). Early word processing programs such as runoff and troff provided markup for appearance, but rapidly added macro facilities that could be used for more content-oriented markup. HTML documents on the World-Wide Web followed something of the reverse of this progression. Early HTML was often marked up by content (title, heading), but as designers sought ways to make their pages look distinctive, markup became more appearance-oriented. The cascading style sheets extension to HTML seeks to support more distinctive graphic design while retaining content-oriented markup; its success remains to be seen. Interaction in HTML remains at a more semantic level, perhaps, because the facilities provided are so limited. This can be a benefit. Imagine downloading an HTML input <form>, turning it into a menu to be filled in using a touch-tone telephone or a Braille teletype, and then sending the resulting data back in HTML, with no need to inform the server as tohow the user filled in the form.

We thus claim that Open Syntax and user interface management system technology can provide a useful technical substrate for improving access by elderly users. Because the same technology is useful for other purposes, such as mobile computing, new forms of multi-mode interaction, and universal access for disabled users, we hope it will be more likely to be adopted. From a practical

viewpoint, tying universal access to a technology with a broader target audience will help to achieve its wider adoption.

5. MODE-INDEPENDENT INTERACTION

The notion of interchangeable front ends for user interaction might also be extended to a more general idea of interaction and knowledge that can appear in different modes as needed. A computer might display information about how to repair a machine or a summary of daily stock market performance in a variety of media. It could present pictures or diagrams, animated video clips, a text document, a spoken lecture or narrative on the subject, or various multi-media combinations of these, such as a diagram with a spoken narration about it. To realize this today, each of the separate representations of, for example, how to repair the machine must have been stored in the computer individually ahead of time. The video clips, the spoken lecture, and the combination description must each be input and stored separately in the computer, to be replayed on command. Some ad-hoc translation from one medium to another may be possible, such as extracting still pictures from a video or creating speech from a text file by speech synthesis. But such translations often result in presentations that are suboptimal in their new media. Information is lost in the translation, and other information that might be more appropriate in the target medium was not present in the source. Instead, we envision a situation where much of the knowledge about how to repair the machine might be stored once, in a form that does not depend on the choice of media used, and then output in different media and forms as needed to suit the individual user and the situation. The user may have personal preferences as to which media or combinations are preferred, or learning styles that work better or worse for different individuals, or disabilities—a visually impaired user might use a spoken or other auditory display instead of a visual one. The playback situation may also require different modes—the user could be sitting in front of and watching a screen or driving a car, in which case an auditory presentation would be preferred. [12]

The parallel between this situation and the UIMS should be obvious. In both cases, there is an underlying core of (media-independent) information or operations that might be expressed in various ways and a separate component that converts that information into a specific presentation or interface to the user. Implicit in both is the notion that the presentation or interface component might be changed, without having to modify the knowledge base or application component in order to provide an alternate view or interface for the same knowledge or application functionality. While there has been some research in the user interface software area working toward systems that can automatically generate a user interface from a specification of the application functionality[1, 2], this is largely a problem for future research. However the UIMS-like framework itself (i.e., the dialogue independent application representation plus separate dialogue component) applies both to automatically-generated interfaces and manually-designed specifically-authored ones.

6. NEXT GENERATION INTERACTION STYLES

This section outlines some more general trends in user interface research. In most cases these are being developed with no thought to their impact on elderly users. We describe them here in order to open them up for consideration of their potential benefits and problems for the elderly.

The first is less a future development than one that has already come to pass—increased reliance on visual communication and interaction. Modern "graphical user interfaces" are indeed highly graphical: they require considerable visual acuity to perceive; good manual dexterity to manipulate a cursor with respect to the displayed graphics; and, because they use visual communication so effectively, they are often difficult to translate into other modes for visually impaired users. Early text-based interaction with teletypes or "glass teletype" displays could easily be translated into spoken words or larger type. Compare this to the screen of a modern GUI. While the words on the screen can be spoken or enlarged, there is much information communicated by the arrangement of objects on the screen and by subtle graphical details. These greatly enhance interaction for many users, but they make it more difficult to provide a translator into a non-visual or less highly-visual mode. This is a well established problem with today's graphical interfaces, perhaps likely to get worse as new interfaces make fuller use of more senses for communication. Direct access to the underlying syntactic-semantic interface would of course provide one solution here, by bypassing the graphics and directly accessing the basic functionality in a mode-independent way.

A related trend is toward increased naturalness in user interfaces. Such interfaces seek to make the user's input actions as close as possible to the user's thoughts that motivated those actions, that is, to reduce the "Gulf of Execution" described by Hutchins, Hollan, and Norman[7], the gap between the user's intentions and the actions necessary to input them into the computer. The motivation for doing this is that it builds on the equipment and skills humans have acquired through evolution and experience and exploits them for communicating with the computer. Direct manipulation interfaces[16] have enjoyed great success, particularly with new users, largely because they draw on analogies to existing human skills (pointing, grabbing, moving objects in space), rather than trained behaviors. Virtual reality interfaces, too, gain their strength by exploiting the user's pre-existing abilities and expectations. Navigating through a conventional computer system requires a set of learned, unnatural commands, such as keywords to be typed in, or function keys to be pressed. Navigating through a virtual reality system exploits the user's existing, natural "navigational commands," such as positioning his or her head and eyes, turning his or her body, or walking toward something of interest. Tangible User Interfaces similarly leverage real-world manipulation of real physical objects to provide a more natural interface[8]. The result is to increase the user-to-computer bandwidth of the interface and to make it more natural, because interacting with it is more like interacting with the rest of the world. Such interface require less memorization of commands, because they leverage things the user already knows. We conjecture that this might be especially significant for elderly users.

Another trend in future interaction is toward lightweight, non-command, passive interactions, which attempt to glean inputs from context and from physiological or behavioral measures. We can thus obtain input from a user without explicit action on his or her part. For example, behavioral measurements can be made from changes the user's typing speed, general response speed, manner of moving the cursor, frequency of low-level errors, or

other patterns of use. A carefully designed user interface could make intelligent use of such information to modify its dialogue with the user, based on, for example, inferences about the user's alertness or expertise (but note that there is also the potential for abuse of this information). These measures do not require additional input devices, but rather gleaning of additional, typically neglected information from the existing input stream. In a similar vein, passive measurements of the user's state may also be made with additional hardware devices. In addition to three-dimensional position tracking and eye tracking, a variety of other physiological characteristics of the user might be measured and the information used to modify the computer's dialogue with its user[15]. Blood pressure, heart rate, respiration rate, eye movement and pupil diameter, and galvanic skin response (the electrical resistance of the skin) are examples of measurements that are relatively easy and comfortable to make, although their accurate instantaneous interpretation within a user-computer dialogue is an open question. A more difficult measure is an electro-encephalogram, although progress has been made in identifying specific evoked potential signals in real time. Looking well beyond the current state of the art, perhaps the final frontier in user input and output devices will be to measure and stimulate neurons directly, rather than relying on the body's transducers. This is unrealistic at present, but it may someday be a primary mode of high-performance user-computer interaction. If we view input in HCI as moving information from the brain of the user into the computer, we can see that all current methods require that this be done through the intermediary of some physical action. We strive to reduce the Gulf of Execution, the gap between what the user is thinking and the physical action he or she must make to communicate that thought. From this point of view, reducing or eliminating the intermediate physical action ought to improve the effectiveness of the communication for all users. The long-term goal might be to see the computer as a sort of mental prosthesis, where the explicit input and output steps vanish and the communication is direct, from brain to computer.

Another way to predict the future of computer input devices is to examine the progression that begins with experimental devices used in the laboratory to measure some physical attribute of a person. As such devices become more robust, they may be used as practical medical instruments outside the laboratory. As they become convenient, non-invasive, and inexpensive, they may find use as future computer input devices. The eye tracker is such an example[11]; the physiological monitoring devices discussed may well also turn out to follow this progression.

In each of these cases, from relatively mundane use of context, behavior and simple passive measurement to notions that sound like science fiction, all of these lightweight or passive interface reduce the effort the user must make to communicate with the computer by taking information from implicit inputs rather than requiring all input to be explicitly produced by the user. We conjecture that this may be a benefit to elderly users with reduced physical or cognitive ability or reduced speed at which they can generate input.

We might also predict the future of user interfaces by looking at some of the characteristics of emerging new computers. The desktop workstation seems to be an artifact of past technology in display devices and in electronic hardware. In the future, it is likely that computers smaller and larger than today's workstation will appear, and the workstation-size machine may disappear. This will be a force driving the design and adoption of future interface mechanisms. Small computers are already appearing—laptop and palmtop machines, personal digital assistants, wearable computers, and the like. These are often intended to blend more closely into the user's other daily activities. They will certainly require smaller devices, and may also require more unobtrusive input/output mechanisms, if they are to be used in settings where the user is simultaneously engaged in other tasks, such as talking to people or repairing a piece of machinery. At the same time, computers will be getting larger. As display technology improves, as more of the tasks one does become computer-based, and as people working in groups use computers for collaborative work, a office-sized computer can be envisioned, with a display that is as large as a desk or wall (and has resolution approaching that of a paper desk). Such a computer leaves considerable freedom for possible input means. If it is a large, fixed installation, then it could accommodate a special-purpose console or “cockpit” for high-performance interaction. It might also be used in a mode where the large display is fixed, but the user or users move about the room, interacting with each other and with other objects in the room. In that case, while the display may be very large, the input devices would be small and mobile. The flexibility gained by this wider range of physical form factors ought to be beneficial for accommodating elderly users, but it has not yet been explored for this purpose.

Two other trends in user interfaces are continuous input and output and parallel interaction across multiple modes[5]. These can be seen clearly in virtual reality interfaces, but the fundamental characteristics are common to a wider range of emerging user interfaces. They share a higher degree of interactivity than previous interfaces. They involve continuous input/output exchanges between user and computer (using gestures and sensors) rather than streams of discrete tokens (using mouse and keyboard). Moreover, their user-computer dialogues often involve several parallel, asynchronous channels or devices, rather than the single-thread nature of todays mouse and keyboard interfaces. These two characteristics will have a considerable effect on user interface software[13], but it is not yet clear whether these will be good, bad, or neutral for elderly users.

7. ACKNOWLEDGMENTS

I want to thank Jim Schmolze for collaborating in our exploration of mode-independent interaction and my colleagues and students in the Tangible Media Group at MIT and the EECS Department at Tufts for all sorts of help and valuable discussion.

Research described in this paper has been supported in part by NSF, NEH, Office of Naval Research, the Berger Family Fund, the Tufts Selective Excellence Fund, and by the Things That Think and Digital Life Consortia of the MIT Media Lab.

8. REFERENCES

[1] C.M. Beshers and S.K. Feiner, “Scope: Automated Generation of Graphical Interfaces,” Proc. ACM UIST'89 Symposium on User Interface Software and Technology, pp. 76-85, Addison-Wesley/ACM Press, Williamsburg, Va., 1989.

[2] J. Foley, W.C. Kim, S. Kovacevic, and K. Murray, “Defining Interfaces at a High Level of Abstraction,” *IEEE Software*, vol. 6, no. 1, pp. 25-32, January 1989.

[3] J.D. Foley and V.L. Wallace, "The Art of Natural Graphic Man-Machine Conversation," *Proceedings of the IEEE*, vol. 62, no. 4, pp. 462-471, 1974.

[4] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes, *Computer Graphics: Principles and Practice,* Addison-Wesley, Reading, Mass., 1990.

[5] M. Green and R.J.K. Jacob, "Software Architectures and Metaphors for Non-WIMP User Interfaces," *Computer Graphics*, vol. 25, no. 3, pp. 229-235, July 1991.

[6] H.R. Hartson and D. Hix, "Human-computer Interface Development: Concepts and Systems for its Management," *Computing Surveys*, vol. 21, no. 1, pp. 5-92, 1989.

[7] E.L. Hutchins, J.D. Hollan, and D.A. Norman, "Direct Manipulation Interfaces," in *User Centered System Design: New Perspectives on Human-computer Interaction*, ed. by D.A. Norman and S.W. Draper, pp. 87-124, Lawrence Erlbaum, Hillsdale, N.J., 1986.

[8] H. Ishii and B. Ullmer, "Tangible Bits: Towards Seamless Interfaces between People, Bits, and Atoms," *Proc. ACM CHI'97 Human Factors in Computing Systems Conference*, pp. 234-241, Addison-Wesley/ACM Press, 1997.

[9] R.J.K. Jacob, "Using Formal Specifications in the Design of a Human-Computer Interface," *Communications of the ACM*, vol. 26, no. 4, pp. 259-264, 1983. Also reprinted in Software Specification Techniques, ed. N. Gehani and A.D. McGettrick, Addison-Wesley, Reading, Mass, 1986, pp. 209-222. http://www.eecs.tufts.edu/~jacob/papers/cacm.txt [ASCII]; http://www.eecs.tufts.edu/~jacob/papers/cacm.ps [Postscript].

[10] R.J.K. Jacob, "An Executable Specification Technique for Describing Human-Computer Interaction," in *Advances in Human-Computer Interaction, Vol. 1*, ed. by H.R. Hartson, pp. 211-242, Ablex Publishing Co., Norwood, N.J., 1985.

[11] R.J.K. Jacob, "Eye Movement-Based Human-Computer Interaction Techniques: Toward Non-Command Interfaces," in *Advances in Human-Computer Interaction, Vol. 4*, ed. by H.R. Hartson and D. Hix, pp. 151-190, Ablex Publishing Co., Norwood, N.J., 1993. http://www.eecs.tufts.edu/~jacob/papers/hartson.txt [ASCII]; http://www.eecs.tufts.edu/~jacob/papers/hartson.ps [Postscript].

[12] R.J.K. Jacob and J.G. Schmolze, "A Human-Computer Interaction Framework for Media-Independent Knowledge," *AAAI Workshop on Representations for Multi-Modal Human-Computer Interaction*, pp. 26-30, Position paper, Technical Report WS-98-09, AAAI Press, Menlo Park, Calif., 1998. http://www.eecs.tufts.edu/~jacob/papers/aaai98.html [HTML]; http://www.eecs.tufts.edu/~jacob/papers/aaai98.ps [Postscript].

[13] R.J.K. Jacob, L. Deligiannidis, and S. Morrison, "A Software Model and Specification Language for Non-WIMP User Interfaces," *ACM Transactions on Computer-Human Interaction*, vol. 6, no. 1, pp. 1-46, March 1999. http://www.eecs.tufts.edu/~jacob/papers/tochi.pmiw.txt [ASCII]; http://www.eecs.tufts.edu/~jacob/papers/tochi.pmiw.ps [Postscript].

[14] D.R. Olsen, *User Interface Management Systems: Models and Algorithms,* Morgan Kaufmann, San Francisco, 1992.

[15] R.W. Picard, *Affective Computing,* MIT Press, Cambridge, Mass., 1997.

[16] B. Shneiderman, "Direct Manipulation: A Step Beyond Programming Languages," *IEEE Computer*, vol. 16, no. 8, pp. 57-69, 1983.

[17] B. Shneiderman, *Designing the User Interface: Strategies for Effective Human-Computer Interaction, Third Edition,* Addison-Wesley, Reading, Mass., 1997.