Behavior Research Methods, Instruments, & Computers
1991, 23 (4), 523-536

NeXT in the psychology laboratory: An example of an auditory pattern tracking task

ANNABEL J. COHEN, MICHAEL P. LAMOUREUX, and DEBORA A. DUNPHY
Dalhousie University, Halifax, Nova Scotia, Canada

The NeXT computer system is a relatively new, inexpensive, 68040-based computer with high computational power, storage, graphics, audio, and other capabilities. The present article examines the effectiveness of the NeXT for studies of auditory sequential memory. In these studies, subjects track the time of occurrence of a sequence of tones, using a mouse that controls the position of sliders represented on the computer screen interface. This interface allows the subject to represent the different sounds on the *y*-axis and the time of their occurrence on the *x*-axis. The computer produces random orders of stimulus sequences, provides feedback, and stores and transforms accuracy and response time data from the slider positions for subsequent analysis. A tutorial for the development of this interface and its variations is provided. As well, procedures for programming the experiment and the operation of the interface are described, accompanied by a frank account of issues surrounding development and management. It is concluded that high-quality sound and ease of both graphics interface design and program modification make NeXT an attractive option for research on auditory sequential order, promising much for further applications as well. Successful use of the NeXT, however, demands much from technical personnel for both programming and system management.

The NeXT computer system, released in September 1988, was designed to serve the university community better than computers such as Macintosh, MS-DOS machines, and even SUN workstations. The present article is the first in *Behavior Research Methods, Instruments, & Computers* to focus on this "new" machine. This long time lag after NeXT's introduction is but one indicator that NeXT has not taken behavioral researchers by storm. Why not? The present article details the experience of one laboratory that gambled on the NeXT for auditory research. In particular, the application of the NeXT for research on sequential memory for auditory patterns is described.

Current models of the NeXT computer system run on the Motorola 68040 25-MHz CPU, which has an integral floating point unit.[1] In its most affordable configuration, the NeXTstation includes 8 MB of main memory, a 105-MB internal hard disk drive, the multitasking UNIX operating system, a 17-in. (diagonally) high-resolution (1,120 × 832 pixel) monochrome display, stereo CD-quality (16-bit) sound synthesis based on the Motorola 56001 DSP, Ethernet support, the sophisticated mathematical software package Mathematica, and Display PostScript. The more costly NeXTcube offers the additional storage option of a 256 MB optical disk drive. Color versions are available for all models. Thompson and Baran (1988), Thompson and Smith (1990), and Webster (1989) provide further details on the earlier models based on the Motorola 68030.

The cost of the NeXTstation ($4,995 US list, with discounts as low as $3,500 through university programs, depending on volume commitment; with laser printer, an additional $1,795 list) compares well with that of Macintosh and IBM microcomputers, both of which lack the storage, computational power, graphics, and sound and communication capabilities offered as standard on the NeXT. Yet although the NeXT offers much more potential for the same cost, there are hidden costs too. Because it is a new machine, little software is available for it; therefore, one needs to produce the requisite software, to translate applications from older systems to the NeXT, and even to learn how to use the NeXT. On such a complex machine, system management requires considerable expertise and time. Thus, in spite of certain superior aspects of the NeXT, older tried and true machines like the IBM (and compatibles) and the Macintosh still offer certain advantages.

In our laboratory, we had been using the Commodore Amiga with considerable success for research on auditory perception (Cohen & Mieszkowski, 1989). The Amiga, however, provided only 8-bit resolution for sound synthesis, and some of our research required greater resolution than that. High-quality sound was a standard feature on the NeXT computer and not on any others. This

This work was supported by an operating grant and an equipment grant from the Natural Sciences and Engineering Research Council of Canada, awarded to A. J. Cohen. We thank M. Mieszkowski for first directing our attention to NeXT; Tracy Rose and Judy Roy from the NeXT Waltham office for cooperation; and Steve Burke, Bradley Frankland, Jim Leary, and Michael McNamara for testing the tutorial and commenting on the paper. Appreciation to the anonymous reviewers is also extended. Requests for reprints may be sent to A. J. Cohen, Department of Psychology, Dalhousie University, Halifax, Nova Scotia B3H 4J1, Canada (e-mail: acohen@AC.DAL.CA).

factor was probably the most important one in our choosing the NeXT. It meant learning about one machine rather than about both a new machine and a new peripheral sound-generating device. We have absolutely no regrets about the choice of the NeXT for this work. We are satisfied with its sound-production capabilities for work in auditory pattern recognition. In addition, other capabilities of the system have facilitated our progress. In particular, the ease of developing graphics for the human–computer interface and the method for relating this interface to software have emerged as very attractive features. However, this progress has depended on high-level technical support for system management and program development.

In what follows, we report some of our experiences of using the NeXT. More specifically, we describe the development of a computer program for studying listeners' abilities to remember the order of a sequence of tones. We first outline the problems of past methods for studying auditory sequential order. We then survey the step-by-step development of the interface and what is involved in making variations on that interface. A detailed tutorial is provided in the Appendix for those who wish immediate, hands-on experience, the best way of getting to know the machine. Subsequently, information about the design of the underlying software is provided. This part of the report has potentially two audiences, the programmer and the researcher director, who may be one and the same person but often are not. The programmer will be able to see what is specifically required in developing software for the experiment with a given interface, and the research director who may be less familiar with programming details will have sufficient information to determine the level of expertise required to take advantage of programming on the NeXT.

RESEARCH METHODS FOR STUDYING AUDITORY SEQUENTIAL MEMORY

The ability to remember sequences of events is a long-standing research interest in experimental psychology. Referred to sometimes as the problem of serial order, it has been addressed with strings of digits and lights in sequence-completion tasks and sequence-reproduction tasks (Restle, 1970). Some researchers have examined the listener's ability to represent the order of a sequence of tones as well. For example, Warren and Byrnes (1975) asked subjects to identify which of six possible orders of four tones had occurred. In another task, their subjects selected four cards (representing the lowest tone, the next lowest, the next lowest, and the highest) in the order in which they thought these tones had appeared in the sequence. To study recall of serial order in longer sequences, Cohen and Frankland (1990) asked subjects to record the order of an 8-tone sequence on an 8 × 8 matrix with a pencil. There was one printed matrix to complete for each of 36 sequences tested. Data were scored by hand. Still other researchers have asked musically trained listeners to use musical notation to represent their memory of sequences (Boltz & Jones, 1986).

The methods for studying auditory sequential memory described above are limited in various ways. Absolute identification as used by Warren and Byrnes (1975) is confined to a small subset of orders, since scanning hundreds of possible patterns for matching is unpractical and memory for more than, say, seven is error prone. Ordering of cards, also used by Warren and Byrnes, is cumbersome and requires that the experimenter write down each order as the subject produces it. Paper-and-pencil tasks, as used by Cohen and Frankland (1990), allow many possibilities for human error: subjects may fill in more than one square at a time, they may forget to complete a square, and there is a subjective element in scoring the data. Musical notation, as used by Boltz and Jones (1986), is limited both to musicians and to tones that fall within the musical notation categories. In addition, none of these methods allows for the collection of response time, which would complement accuracy data. A method was therefore devised on the NeXT to overcome these problems. The particular experiments to be described examined subjects' ability to remember or track the order of presentation of eight tones.

An interface on the screen of the NeXT monitor was defined with eight vertical sliders, one for each serial position in the eight-tone sequence (see Figure 1). The first slider represented the first temporally presented tone of the sequence, the second slider represented the second tone, ... the eighth slider represented the final tone. The vertical axis represented the ordinal position of the tone with respect to highness and lowness in pitch. Only eight vertical positions were represented, because for any sequence of eight different tones, there are only eight ordinal positions necessary to describe the set. In the experiment, the subject heard five repetitions of a sequence of eight tones and was required to indicate for each serial position which particular tone had occurred. Thus, to represent an ascending scale of eight tones, successive sliders would occupy Positions 1, 2, 3, 4, 5, 6, 7, and 8. The positioning of the sliders was controlled with a mouse. Placement of the cursor at a potential position for the slider and clicking the mouse automatically attracted the slider to that position.

Tones in these studies ranged from 200 to 400 msec in duration, including intertone interval. Thus sequences were between 1.6 and 3.2 sec in length. Sequences were repeated up to five times in succession. After the subject had completed his or her response and the Done button was pressed, the position of the sliders was recorded by the computer and feedback regarding the number of sliders correct was returned to the subject. The time from the onset of the auditory pattern to the end of the response was also recorded. The subject, when ready, pressed the Go button, and the next trial began.

This interface provided a number of advantages over earlier methods of examining auditory pattern tracking.

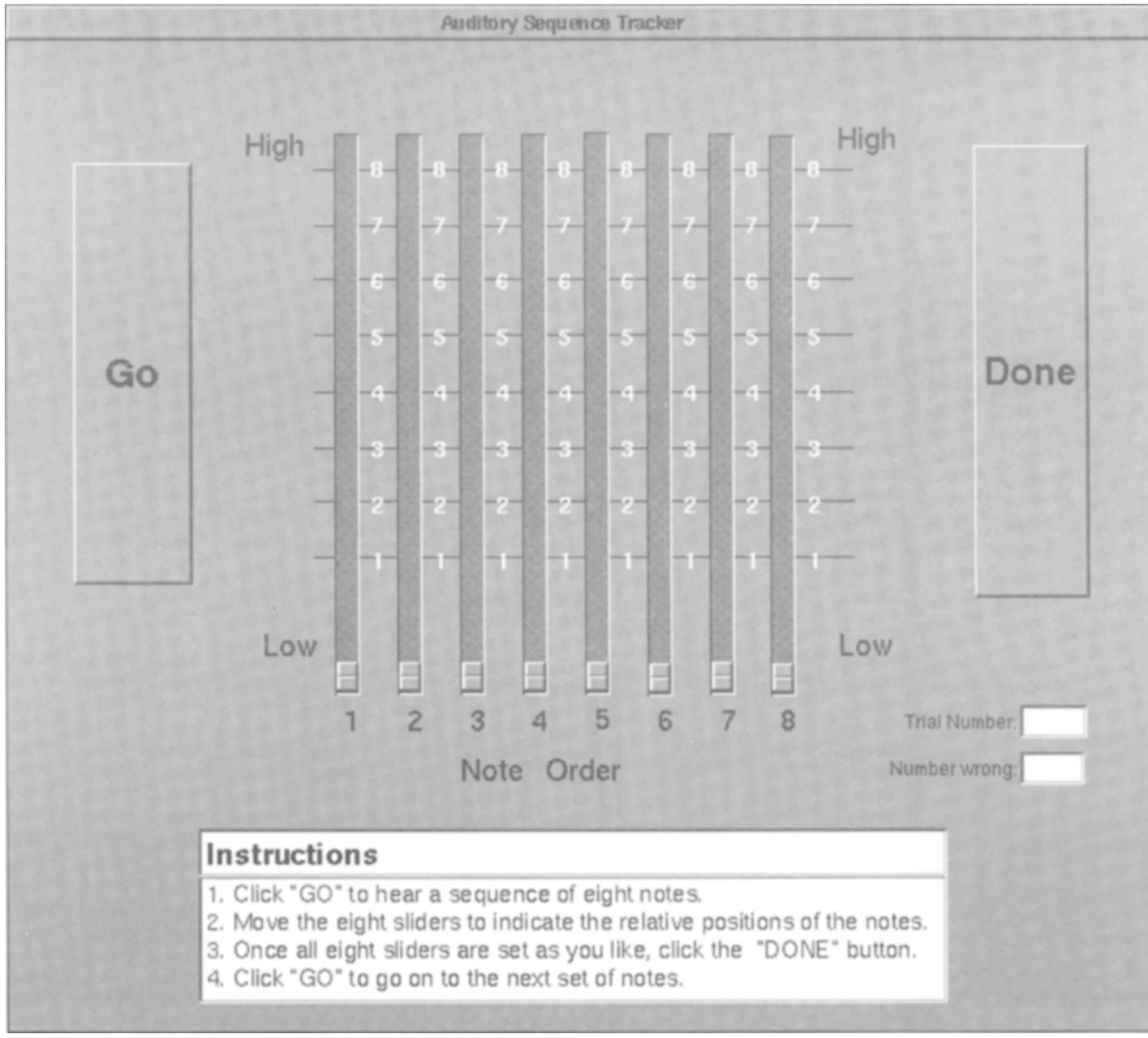


Figure 1. Test window for the Auditory Sequence Tracker Test.

In contrast to the paper-and-pencil task of Cohen and Frankland (1990), there was a fresh panel presented for each new sequence; thus, judgments were independent. Sequence order was random and different for each subject. Only one position could be filled in for each serial position. The data from the sliders were scored by the computer; hence no experimental error entered into the results. The data were also automatically transformed and sent to another computer for analysis, thus saving time and increasing efficiency. The method provided a second dependent measure, response time. Feedback was provided to subjects. The technique saved material resources such as paper (to do the test properly would have involved over 70 separate sheets per subject). Finally, the interface was amenable to a variety of changes that could be made to examine theoretical and practical issues.

These advantages were thought to outweigh certain disadvantages. For example, subjects were not equally familiar or facile with the mouse. Thus, additional research would be required to distinguish aspects of performance associated with the input method itself from aspects associated with the variables of interest in the task, scale structure, and serial rules, although the NeXT did make it possible to develop these additional control studies. The computer and even the optical disks were expensive relative to a cassette tape recorder and a few pencils. Testing was limited to 1 subject per machine. Such costs might be balanced against those of personnel costs incurred by time spent to hand code the data.

In what follows, we first provide an overview of the development of the interface and its variations. Then we describe what is involved in developing the program to

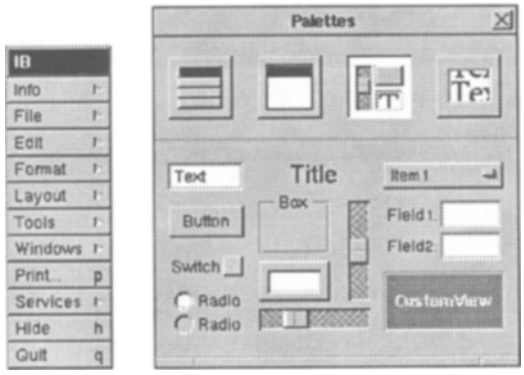


Figure 2. (a) Main menu for the Interface Builder. (b) Interface Builder's Palettes window.

link the interface to the computer. In our focus on this particular interface and paradigm, we do not mean to imply that it is the only or best way to study auditory pattern tracking. Our purpose is to highlight a technique that is readily available on the NeXT and is not as available on other machines. It is an example of new paradigms that can be invented and explored in ways that may not be accessible on other machines at present.

BUILDING AN INTERFACE: TUTORIAL

The NeXT Interface Builder enables one to create a user interface directly with graphics rather than indirectly with files of programming code. From a palette of predefined objects, such as sliders, windows, buttons, switches, or objects of your own design, the interface takes shape as easily as moving the mouse. The interface is created by dragging objects from a palette (Figure 2b) and physically manipulating their features, such as appearance and position. A person with no previous experience with NeXT might require from 1 to 2 h to produce the interface shown in Figure 1; the experienced user could require less than 20 min. A complete tutorial for the development of the interface in Figure 1 and its transformation into Figure 3 is provided in the Appendix. Here, we provide only an overview of the procedures but emphasize that the best way to understand the graphical user interface (GUI) is first-hand experience.

Launching the Interface Builder will automatically generate the application menu (IB menu, Figure 2a) and the Palettes window (Figure 2b); a command from the menu will open an empty window and thus display the

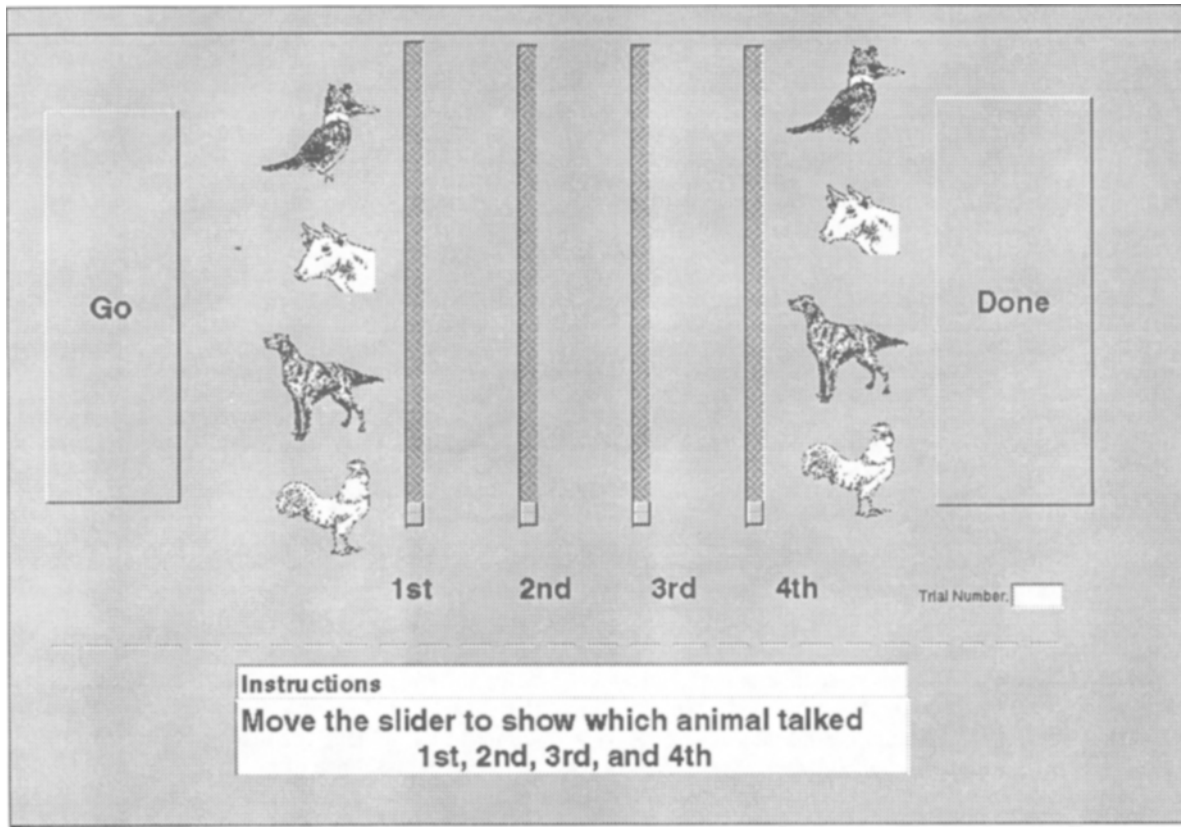


Figure 3. Interface adaptation to the test window of the Auditory Sequence Tracker Test. Animal pictures represent four different sounds and replace the eight different auditory frequencies of the original version, in Figure 1.

foundation for interface building. The application is constructed by dragging objects from the Palettes window to the empty window, where the object may be directly manipulated (title, size, position, etc.) with the use of menu commands and the mouse. Whereas dragging the cursor on an object or window will vary its size and position, the Inspector command (on the Tools submenu; Figure 2a) enables one to alter graphic and nongraphic attributes of windows or objects within windows more precisely. In the present case, Go/Done buttons and sliders were created with the aid of menu commands that facilitate either the creation of replicas or the alignment of a multitude of objects or patterns. Even without any underlying programming, the response of buttons and sliders can be tested by using the Test Interface command available on the Files submenu of the Interface Builder menu (Figure 2a). Thus, in this test mode, clicking on the Go button will highlight it, and clicking and moving the cursor over a slider will move the slider knob.

Changes to the interface for a different application can be relatively simple. For example, through the alteration of button sizes, eliminating sliders, changing instructions, and adding icons, the Auditory Sequence Tracker can now serve a completely different purpose. For example, the Auditory Sequence Tracker was modified for testing children's ability to report the order of four animal vocalizations. The interface shown in Figure 3 was developed.

The task for the child is basically the same as that for the adults, but the child must report the order of four different animal sounds rather than eight different frequencies. This particular adaptation also illustrates the use of the built-in graphic resources of the Digital Webster dictionary, the source of the pictures of the animals. Another response interface, shown in Figure 4, was used to examine the listeners' ability to match a graphic representation to the auditory sequential pattern, and the interface shown in Figure 5 was used to examine numerical responses to the sequence. We emphasize that construction of such interfaces requires a matter of minutes, and minor modifications, a matter of seconds. NeXT promotion materials offer many examples of more complex and impressive graphics than these. Our examples illustrate specific applications in our laboratory. Having now suggested that such interfaces can be relatively easily designed, we next address the question of the ease of their implementation.

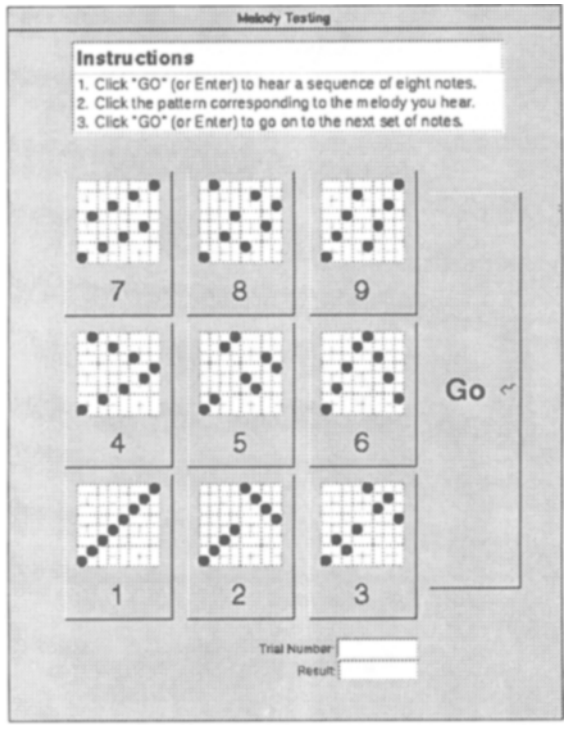


Figure 4. Interface for testing pattern matching of auditory sequences.

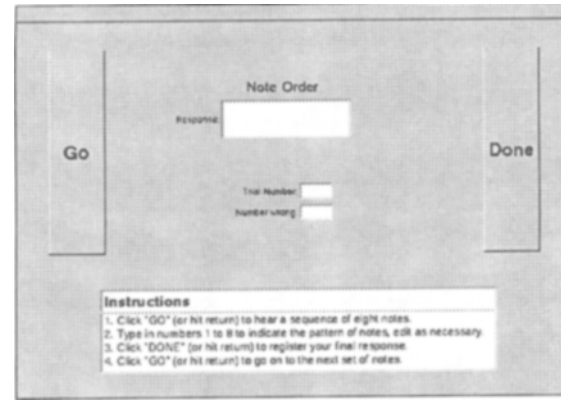


Figure 5. Interface for testing auditory sequence memory through numeric response.

SOFTWARE ISSUES: PROGRAM DEVELOPMENT

The source program for the Auditory Sequence Tracker consists of about 700 lines of Objective C code, of which approximately 40% were automatically generated by the Interface Builder, and the remainder developed over about 40 h by an experienced C programmer (but not NeXT experienced). What is remarkable is that although the user interface for the program is quite complex, involving windows, buttons, sliders, and editable text fields, no code is required to control these graphical objects. In fact, once the user interface has been designed (by dragging and placing objects on the screen using the Interface Builder, as described in the last section), the program can be immediately compiled into a functional shell of an application. It is the NeXTStep environment that takes care of the user interface and makes programming GUI-based applications on the NeXT relatively effortless. NeXT provides a number of tutorial programs on the system that demonstrate the basics of writing a graphically based application in Objective C; more complex examples are available from archive sites on the Internet.

Objective C is used in order to communicate with the graphical objects in the user interface; the standard GNU C

compiler that comes with the NeXT System 2.0 has been extended to compile standard C, Objective C, or C++ files. It is neither difficult to make the transition to an object-oriented mode of programming, nor is it even necessary, because it is possible first to write the major part of a program using the usual procedural methods of standard C, and then to exploit Objective C to communicate with the user interface.

To communicate with the GUI, it helps to think of the objects on the screen as being smart, as if they know how to do things themselves; all you need to do is tell them what to do for you. For instance, to tell a sliding control to set itself to Position 2.5 on the screen, the code is:

```
[theSlider setFloatValue:2.5];
```

This bracketed notation is the Objective C syntax for sending the message setFloatValue to the object theSlider, with parameter 2.5. This action will cause the slider to move its control to the position corresponding to value 2.5—it moves on the screen and updates values automatically.

Other collections of data can be treated as objects and manipulated via Objective C. For instance, the Sound Kit provides simple methods for working with sounds. To load a sound file from disk and play it, one creates a sound object, sends it a message to load in the file, followed by another message to play. This takes only four lines of code in Objective C:

```
id theSound;                              // define a variable
theSound = [Sound new];                   // create sound obj
[theSound readSoundFile:"foo.snd"];       // load in data file
[theSound play];                          // play it
```

There are also object-based methods for combining sounds, which are used in our programs to create melodies from collections of short sound files representing tones. Disk access can also be accomplished from an object-oriented perspective, but in this application, the standard C library functions fopen, fclosed, fread, and fwrite were used.

It is also possible to modify given objects to create new ones that are similar to the original. For instance, the Application object of the NeXTStep kit was modified so that when our test program is launched from a shell, its front window would immediately be activated, with a cursor ready and waiting for text entry into a text form. Thus, subjects in an experiment would be presented with a test program ready for their input. In the Interface Builder, using the Class List panel, a subclass of the Application object was created, named AutoStart. In the Class Inspector, the appDidInit action was added, and theFrontWindow and theFrontForm outlets then unparsed this object. This created two Objective C source files, AutoStart.h and AutoStart.m, to which the addition of three more lines of code yielded the following:

```
// file AutoStart.h
// Generated by Interface Builder
#import <appkit/Application.h>

@interface AutoStart:Application
{
    id theFrontWindow;
    id theFrontForm;
}
- appDidInit:sender;
@end

// file AutoStart.m
// mostly generated by Interface Builder
#import "AutoStart.h"
@implementation AutoStart
-appDidInit
{
    [self activateSelf:YES];                              // added
    [theFrontWindow makeKeyAndOrderFront:self] // added
    [theFrontForm selectTextAt:0];                        // added
    return self;
}
@end
```

The default File's Owner object in the icon list was then reclassed as an AutoStart object, and connections were drawn between this object and the FrontWindow and FrontForm objects. Thus the startup behavior of our application was modified.

Since the Auditory Sequence Tracker is a relatively complicated program, the programmer code was partitioned into four objects: the Rounder, the SoundBag, the ScoreKeeper, and the Brains. The Rounder simply monitors the sliders in the graphical interface, and forces them to move only to integer step values 1 to 8, so that the subject's response is unambiguous. The SoundBag keeps track of the collection of scales and orders used to create the melodies, in random order, and plays them on request. The ScoreKeeper records to disk the subject's responses in the test, along with a record of the stimulus and timing, and presents feedback to the subject. The Brains is the core of the program, keeping track of timing, noting which melodies have been played so that they are never repeated, waiting for the subject to respond, and basically coordinating all aspects of the test. In the implementation of these modules as Objective C objects, these four not only communicate with the graphical objects in the user interface, but with each other as well, using links and connections created in the Interface Builder.

Figure 6 shows the collection of objects that are active in the program, both user interface objects and the control objects mentioned above, the links between them, and the flow of data. The basic operation is as follows. When the Go button is pressed, a message is sent to the Brains, telling it to start a trial. The Brains passes a message to the SoundBag, telling it to play the next melody, which has been constructed according to rules given in the Configuration File. While the SoundBag is playing the melody, the subject is moving the sliders in the graphical interface. When the Done button is pressed, the Brains gets the message to stop the trial. The Brains then tells

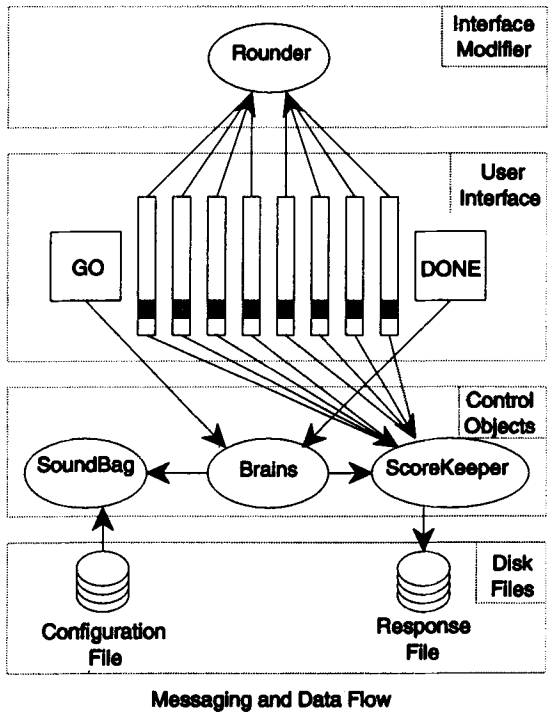


Figure 6. Diagram of Messaging and Data Flow among the Interface Modifier, User Interface, Control Objects, and Disk Files.

the ScoreKeeper to record the position of the sliders and save the response set onto the disk file. The program is now ready for the next trial, so the Brains reinitializes, and waits for the Go button to be pressed again.

In this application, sounds are managed as objects, and creating and playing a melody in the program is thus simply a matter of loading in the separate notes as objects from sound files on disk, combining them to create the melody, and sending a Play command to the combined object. To create the sound files in the first place, NeXT provides two tools: the Sound Kit and the Music Kit (Jaffe & Boynton, 1989). The Sound Kit is the simpler of the two. It is a collection of objects that generate and manage sounds on the NeXT, making use of the CD-quality audio output to produce the sounds, and the toll quality microphone input to record them. With the SoundEditor program, an application provided by NeXT which uses the Sound Kit, a tone can be recorded via the microphone and saved to disk as an object for later use in the melody. Also with the SoundEditor, the waveform of the note can be viewed, and simple modifications such as cutting, inserting, and deleting segments of sound can be made. The quality of the standard microphone input is not optimal, with 8-bit, 8-kHz sampling, but it is useful for recording environmental "notes" such as a dog bark. For higher quality recording, 16-bit digital microphones are available from several third-party suppliers.

For more structured sounds, the Music Kit provides the tools to create tones with specific spectral and temporal characteristics. A simple text file with a list of note frequencies, amplitudes, envelopes, and overtones is the input to a score-playing program, which takes the text and creates the sampled sound as indicated. The sound is synthesized with the DSP hardware on the NeXT and saved to disk for later use by the SoundEditor, in preparation for the melody tester. The Music Kit may also be used to generate sound directly for playback through the NeXT's speaker or line-outs, rather than first saving to disk. The DSP synthesis is fast enough to generate moderately complex sounds in real time, which can be used directly in a program, without the need to manage large sound files in memory or on disk.

If none of these methods is adequate, the samples for a digitized sound can be calculated directly from a mathematical formula within a C program. This is a very flexible way of generating sound for playback, but it is often unnecessary, because the Music Kit alone provides a wide range of sounds. Our choice was to use the Music Kit, saving the sound files to disk for later use in the Auditory Sequence Tracker program. Although very complex sounds may be created in this manner, we generated sine waves in the western chromatic scale, using an exponential ramp up / ramp down envelope. The note parameters to the Music Kit were specified with double precision, generated on the 24-bit Motorola DSP at 44.1-kHz sampling, and stored as 16-bit stereo samples at 44.1 kHz. The sine waves were generated from a 256-sample, 24-bit table by using interpolation, which gave a theoretical total harmonic distortion of less than $1.5 \cdot 10^{-9}$ in the resulting signal.

HARDWARE ISSUES: SOUND AND TIMING

The accuracy of our method of sequence generation is influenced directly by the NeXT hardware in two important aspects, the quality of the sound and the accuracy of the timing. We made extensive measurements of the sound-output capabilities of the NeXT cube (Lamoureux & Cohen, 1990) and found it to be exceptionally good. The sound output is based on a 16-bit, 44.1-kHz digital-to-analog (DAC) converter housed in the video monitor. We measured approximately 13 bits of linearity on the DAC, with a frequency response of $\pm .5$ dB in the range of 1 Hz to 20 kHz, a signal-to-noise ratio of 88 dB, and a uniform phase delay of 6 μsec between the left and right channels. As a pure-tone generator, the NeXT produces a tone whose frequency is accurate to within $\pm .002\%$ (crystal controlled) with harmonics that appear at 75 dB below the fundamental. The DSP56001 signal-processing chip and Music Kit software generate signals of equal or greater quality than what the hardware can reproduce.[2]

The timing accuracy is a much more difficult issue. The NeXT runs on the Mach operating system, a multiuser, multithreading UNIX kernel, which implies that more than one process at a time can be executing on the CPU, even within the same program. Ping commands are available

to coordinate the display of data on the screen with the playing of sounds and the actions of the user (keypressing, mouse clicks), but in principle, the concept of referencing a unique "time" in such a system is complex at best. At the core of the hardware is a real-time clock that can be accessed directly from a test program for microsecond accuracy (although there is at least a 100-μsec overhead in the calling subroutine), so it seems that there is potential for very accurate timing. There is also some suggestion from the NeXT documentation that the DSP56001 can be used directly as a real-time clock and to coordinate the many timed processes in a test, although how to do this is unclear and would require significant programming. Another possibility is to record user events (mouse clicks, keypresses) on the basis of the event record, which store the time accurate to 1/70 of a second. In our case, we recorded the time on the basis of when the user clicked the Done button. Since the GUI records a button activation when the mouse button is *released* and not when it is first clicked, we had a 100-200 msec discrepancy between when the user was done and when the program recorded the time. This discrepancy swamps any other possible timing inaccuracies in the system, but since each timed task took around 10-30 sec, an accuracy of ¼ sec was judged acceptable. Another issue is the question of how a multiuser environment might interfere with the timing of an experiment; that is, multiple users compete for CPU and disk resources, possibly disturbing critical timing. We avoided this problem altogether by running our tests in a single-user mode. These timing problems are not unique to the NeXT; related problems occur with the Macintosh (Keiley & Higgins, 1989; Westall, Perkey, & Chute, 1989) and the IBM PC (Heathcote, 1988).

DEVELOPMENT ISSUES: ADAPTING THE PROGRAM, DEBUGGING, DOCUMENTATION

Once the Auditory Sequence Tracker was up and running, it was a simple matter to modify the program to perform a variety of similar auditory tests. For instance, a small change in the code for Brains added the ability to do blocked tests, in which only one scale was presented during a block of trials. Working with different melodies (including different sequential orders of notes and different scales) was accomplished by changing only the configuration files. We were able to create an entirely new program, to measure the identification of sequences of animal sounds, simply by modifying both the graphical interface and the configuration files. This new program presents the subject with a series of random sequences made up of four animal sounds (bark, moo, cock-a-doodle-do, and tweet), and it prompts the subject to identify the sequence by moving sliders on the screen to match a picture of the animal to the sound (Figure 3). To create this new program, pictures of the animals were added to the graphical interface, four sliders removed (leaving four on the screen, one for each sound in the sequence), and the text changed to describe the new test. The configuration files were changed to specify the new collection of animal sounds. It is significant that this new program was created without modifying a single line of code and was thus quite quickly completed.

Although much of the program development is done within the Interface Builder, the source code is written in Edit, a simple text editor. While developing the program, necessary debugging was accomplished by liberally sprinkling our code with printf statements, enabling observation of the progress of the program as it ran in a UNIX shell. NeXT includes GDB, the GNU source-level debugger, as part of the standard programming environment, although effort was not made to learn it. The actual compile and linking process is automated in the Interface Builder, so that a single menu command updates a make file and runs the compiler and linker, pulling together all the relevant pieces of code. Finished programs tend to be large; 300K is not unusual for even a simple application.

Most but not all of the documentation necessary for programming the NeXT is provided on line; a hard-copy version is much harder to obtain, unless one is willing to print the entire unbound manual. Months passed before the prerelease version of the System 1.0 documentation arrived, but once it came, programming productivity markedly increased. With the introduction of the new System 2.0, we are once again without the appropriate programming manuals.

SYSTEM ADMINISTRATION: MANAGING THE COMPUTER

The effort in maintaining and operating the NeXT computer has significantly affected the development of our programs for auditory research. Although most personal computers (MS-DOS, Macintosh) require little maintenance beyond keeping the hard disk organized, our NeXT required almost constant attention to keep the system running smoothly, occupying a large proportion of the time of the main programmer. The difficulties stem from the multiuser, multitasking UNIX system; such a system is normally considered complex enough to warrant hiring one or two system administrators, as is the case with VAX and SUN mainframes on our campus. Having no choice but to do the work ourselves meant ultimately that a great amount of time was spent learning about UNIX.

The biggest concern is managing programs and data on the hard disk. Although our system has 330 MB of storage, we typically operate with only 25 MB of free space. Approximately 240 MB are permanently occupied by the UNIX operating system, on-line documentation, and NeXT applications, while the rest contains essential data and programs developed for our research. Other large files are stored on removable optical disks. The consequence is that although the system appears to have con-

siderable hard disk space, in fact the disk is almost always nearly filled to capacity. With four active users, the hard disk would often be filled, preventing the saving of files until considerable time was spent cleaning up. Bugs in the system exacerbate the situation: the swap file, part of the virtual memory system, can grow without bound, thereby depleting hard disk space and eventually causing a system crash, usually destroying data in the process. (This bug is related to a memory leak in the Sound Kit, which has *not* been fixed in System 2.0.) Similarly, when a program crashes, the system often creates a core dump, a 4-MB file that also consumes hard disk space and is of no use; we routinely delete these.

Backing up the hard disk onto opticals consumes considerable time, and since one is often prompted by the system during backup, it is necessary to attend to the console throughout the entire process. Apparently there are UNIX commands to facilitate the backup (System 2.0 documents this in the *Network and System Administration* manual), but corresponding file-retrieval procedures were judged to be too difficult for our average users. Much more serious is the inability to back up optical disks. Because each optical disk holds 256 MB, there is too much data to copy the entire disk into the hard disk's free space, and there appears to be no way to copy data directly from one optical disk onto another. Because we store some raw data directly onto optical disks, we settle for backing up only essential files. Incidentally, optical disks *do* crash, and a call to NeXT technical support was necessary to recover the data the several times that this happened. No data were lost, but much time was expended recovering files. No optical crashes have occurred, in our lab, under System 2.0.

Another large task was to establish data links to other computers on campus, including asynchronous serial links to a SUN mainframe (for data analyses) and a local IBM PC, and to establish an Ethernet connection to the campus network and worldwide Internet. We also use Ethernet to link our two NeXT systems together in order to share files and establish network-wide user accounts as described in the NeXT *System Administration Manuals*. This enables us to move data quite effortlessly now, but it did take much delving into the depths of UNIX and again took time away from other research. Moreover, the network solution is not perfect; for instance, we have not found an easy way to mount an optical disk on one NeXT so that it is accessible on a remote machine, despite calls to and a visit from our NeXT representative.

Keeping track of four users on our system, regulating their disk usage, maintaining some security between their accounts, managing commonly accessed programs and files, and educating the users on how to take best advantage of the computer also takes time. There is also a feeling that much is left undone—for instance, backups should be automatic, and some cron processes that execute automatically are probably building large files about which we have no knowledge. Nevertheless, access to the power of a UNIX system is attractive in many ways.

CONCLUSION

Our focus on the Auditory Sequence Tracker represents but one of many possible applications of the NeXT for research in human information processing. Moreover, for just this one application, the NeXT has opened up future directions for research of both theoretical and practical interest. For example, easy alteration of the parameters in the basic task has permitted us to explore theoretical issues concerning structure of the sets or scales of tones in the sequence, as well as the tempo of the tones (Cohen, Frankland, Lamoureux, & Dunphy, 1990). Translation of tones for environmental sounds represented by pictures has enabled developmental studies with young children. Modality specificity of the phenomena we have observed can be examined by using visually presented digits rather than tones. Transformations of the two-dimensional interface to various unidimensional response panels allows us to consider the role of the design of interfaces in tracking tasks. The NeXT, by making a variety of paradigms available, enables us to discover the shared or converging operations underlying an aspect of perception or cognition and to separate out the effects that reflect the particular method used. Many of these types of issues would be markedly more difficult to address with other software/hardware configurations, and—we emphasize—CD quality is standard on the NeXT, which is an important consideration for auditory research. As it stands, it seems that the NeXT can well support a complete research program devoted to the understanding of auditory pattern sequence perception and promises much for other areas as well. Nevertheless, as indicated by the description of development and management in this one instance, the availability of high-level support personnel is essential at least in these initial stages of NeXT's history. This is no doubt why NeXT has linked a support program with sales on university campuses. In many cases, by the time behavioral researchers read this article, such support programs may be well in place for fruitful exploitation of NeXT's potential.

REFERENCES

Boltz, M., & Jones, M. R. (1986). Does rule recursion make melodies easier to reproduce? If not, what does? *Cognitive Psychology*, **18**, 389-431.

Cohen, A. J., & Frankland, B. (1990). Scale and serial order information in melodic perception: Independence or interdependence? *Canadian Acoustics*, **18**, 2-10.

Cohen, A. J., Frankland, B., Lamoureux, M., & Dunphy, D. (1990). Effects of tone sets and serial order on melodic tracking. *Journal of the Acoustical Society of America*, **88**, S91. (Abstract)

Cohen, A. J., & Mieszkowski, M. (1989). Frequency synthesis with the Commodore Amiga for research on perception and memory of pitch. *Behavior Research Methods, Instruments, & Computers*, **21**, 623-626.

Heathcote, A. (1988). Screen control and timing routines for the IBM microcomputer family using a high-level language. *Behavior Research Methods, Instruments, & Computers*, **20**, 289-297.

JAFFE, D., & BOYNTON, L. (1989). An overview of the Sound and Music Kits for the NeXT computer. *Computer Music Journal*, **13**, 48-55.
KEILEY, J. M., & HIGGINS, T. (1989). Precision timing options for the Apple Macintosh family of computers. *Behavior Research Methods, Instruments, & Computers*, **21**, 259-264.
LAMOUREUX, M., & COHEN, A. J. (1990). Evaluation of the NeXT computer system for auditory research. *Journal of the Acoustical Society of America*, **88**, S92. (Abstract)
RESTLE, F. (1970). Serial pattern learning. *Journal of Experimental Psychology*, **83**, 120-125.
THOMPSON, T., & BARAN, N. (1988). The NeXT computer. *Byte*, **13**, 102-114.
THOMPSON, T., & SMITH, B. (1990). Sizing up the cube. *Byte*, **15**, 169-176.
WARREN, R. M., & BYRNES, D. L. (1975). Temporal discrimination of recycled tonal sequences: Pattern matching and naming of order by untrained listeners. *Perception & Psychophysics*, **18**, 273-280.
WEBSTER, B. F. (1989). *The NeXT book*. New York: Addison-Wesley.
WESTALL, R. F., PERKEY, M. N., & CHUTE, D. L. (1989). Millisecond timing on the Apple Macintosh: Updating Drexel's MilliTimer. *Behavior Research Methods, Instruments, & Computers*, **21**, 540-547.

NOTES

1. The upgrade based on the Motorola 68040 microprocessor was available in October 1990. Its floating-point performance is claimed to be up to 10 times faster than the 68030's.

2. We suspect that the new systems released by NeXT use identical sound hardware and would thus have equivalent specifications, but we have not yet measured them. Version 2.1 of the operating system has also been released.

APPENDIX

The following instructions describe how to build the interface for the Auditory Sequence Tracker (Figure 1) and, subsequently, how to change it for a different subject population or different research issue. This tutorial is clearer if one constantly refers to Figure 1 and occasionally glances back to previous instructions and commands should a subsequent section become unclear.

We include the tutorial in full detail for a number of reasons. First, a reader of this article may have access to a NeXT machine but no access to a manual. The availability of this tutorial will provide a means of examining the Interface Builder immediately. Second, the full detail reflects the true picture of just how easy or difficult the task is. We might have left out instructions on how to produce the numbers for slider positions, for example. On the surface, these look no more difficult to produce than the identifications for buttons. This is not the case, and it is well for the prospective user to know it. By no means, however, do we aim to provide a general tutorial with the present example.

Launching the Interface Builder from System Release 2.0 Extended

Assuming you have just logged on to the NeXT computer (System Release 2.0 Extended), the File Viewer, the main directory for the NeXT, will appear in the center of your screen. To launch the Interface Builder, select NextApps (which contains the NeXT applications available to your system) from the left column of the File Viewer. Then, from the adjacent subdirectory of application names, double click on the application name Interface Builder.

Getting Started

Once the Interface Builder is launched, the IB menu will appear to the left (Figure 2a) along with the Palettes window to the right (Figure 2b). The IB menu provides a list of commands and options, and the Palettes window contains predefined objects from which the interface can be built.

Begin a new application by choosing File from the IB menu. From the submenu which now appears, choose New Application. A menu and two windows will appear. The empty window entitled MyWindow appears at the top center of the screen. MyWindow is the interface foundation and Palettes provides the building blocks. At the bottom left is a window entitled UNTITLED followed by your user name (also called file window; see Figure A1). UNTITLED is the default name of the application and thus the application menu until the file is saved (at which time both will assume the name of the saved file). Between the two windows is an untitled menu that will become the main menu for your finished application.

The File window contains seven icons, two of which represent the menu for the application (MainMenu) and the standard empty window (MyWindow, referred to here also as the main window), respectively. Of the remaining icons in this window, Icons and Sounds allow for the addition of symbols or drawings (.tiff or .eps) and the addition of sound files (.snd). File's Owner, Classes, and First Responder are not really utilized in this interface and therefore will not be referred to; for present purposes, only MyWindow and Icons will be discussed.

Since many applications can run simultaneously on the NeXT, the screen can at times become cluttered with open windows; thus, changing applications can result from simply clicking on a window belonging to another application. Therefore, to ensure that the right application is activated, refer to the title of the menu in the upper left corner. For example, the menu now should read "IB". Click anywhere on the File Viewer panel (the main directory for NeXT) and see that the menu title now reads Workspace. To return to the Interface Builder, click any-

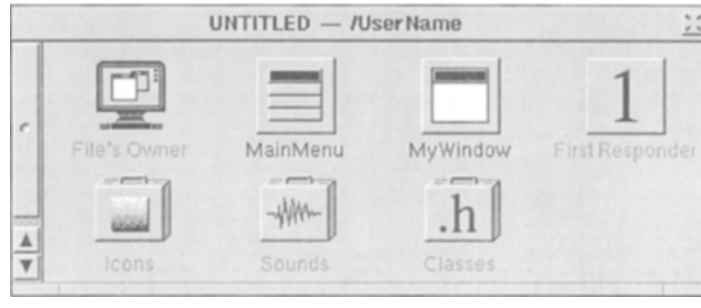


Figure A1. File's Owner window.

where on the MyWindow window, and it will become the foreground work area. The File Viewer will move behind the main window until it is selected again for the foreground. Note that the MyWindow title also belongs to a small icon in the File window. They are essentially one and the same; that is, the one is a miniaturized version of the other. Double clicking on this icon will activate the window or bring it to the forefront.

Resizing "MyWindow"

To start building the Auditory Sequence Tracker, we must first resize the MyWindow to allow for a larger interface. This may be accomplished by *dragsizing* the bottom corner of the window to the appropriate *x,y* coordinates (a marquee will appear to aid in resizing). However, exact alterations of dimensions can be more precisely performed by selecting Inspector under the IB submenu of Tools. The Inspector window appears displaying the attributes of the window (Figure A2, panel a). Note that the top bar of the Inspector window is black (an indicator that the Inspector has been clicked or selected to be active) while the bar on MyWindow is now a dark gray. This color change to dark gray indicates which panel or window the Inspector is presently inspecting. The Inspector's title reads Window Inspector, indicating what item is being inspected. If a button had been the object selected for inspection, the title would read Button Inspector. To select any other window or object for inspection, simply click on it and the Inspector will change accordingly.

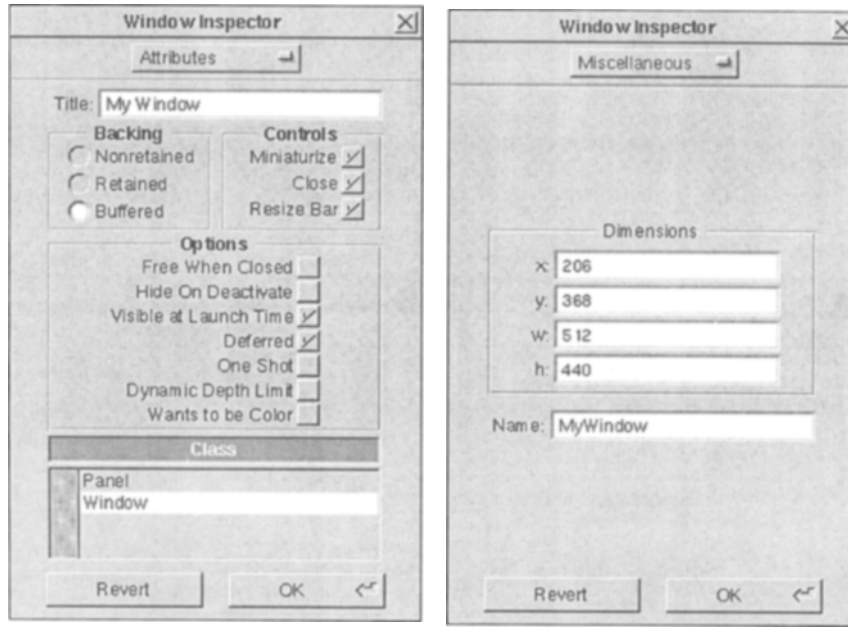
Attributes, the present submenu in the Inspector, alters graphic and nongraphic attributes, such as a button's title alignment or a slider's maximum and minimum values. Using the Window Inspector's controls and options available under Attributes, we can change how the window will respond during the running of the program and its title. At this point, change the name that appears on the title bar from MyWindow to Auditory Sequence Tracker. This change is made by typing in the new title over the highlighted title My Window; clicking the OK button will execute the title change. Under Controls, click the Miniaturize, Close, and Resize Bar switches so that they are *not* check-marked on. Turning these switches off protects inadvertent manipulations of the interface (such as resizing or closing) by subjects in the experiment. Click the OK button to register the changes.

Click on Attributes and a pop-up menu will appear containing other Inspector submenus—Connections, Autosizing, Miscellaneous, Class, and Project. While still depressing the mouse button, drag the cursor down to rest on the Miscellaneous button and release. The Miscellaneous function (Figure A2, panel b) allows for changes to the numerical coordinates of objects or windows, with x and y referring to the position of the object with respect to the main window (or computer screen, depending on the nature of the object) and w and h referring to width and height of the object or window. Set the position at $x = 170$, $y = 55$, and the window's size at $w = 741$, $h = 637$ (in this case, the window's position refers to its position within the boundaries of the screen). When OK is clicked or the Return key is pressed, the window will adjust to match the new coordinates.

Adding Objects

The interface is built from an alphabet of objects in the Palettes window (see Figure 2b). Adding objects to the application is the easiest part of creating the interface. The desired object is simply dragged from the Palettes window to the preferred position on your application.

Creating the "Go" and "Done" buttons. Select the Palettes window to be the front window and drag over a button. (Note that the Palettes window's top bar does not turn black. Since this window is used only as an object depot, it need not be activated to be utilized.) Dragging is done by clicking on the desired



a) b)

Figure A2. (a) Attributes option of the Window Inspector. (b) Miscellaneous option of the Window Inspector.

object in the Palettes window (in this instance, the button) and moving it (while the mouse button is depressed) to the main window; to anchor the object, release the mouse button. The button object is surrounded by tiny squares. These squares or control points, which are only visible when the object is selected, allow for manipulation of the object. (Note that the Window Inspector is now entitled Button Inspector.) In order for the object to no longer be selected, click elsewhere on the window (this selects the window, and the Inspector is once again the Window Inspector). Click again on the button and then on any one of the button's control points so that it turns white, and drag it to the desired size as shown in Figure 1. If the square does not turn white, you have missed grabbing the control point and as a result will only move the button to a new location. With the button still selected, refer to the Inspector's Miscellaneous submenu for a more precise resizing or replacement. The size and position of the button selected for the Auditory Sequence Tracker were $x = 40$, $y = 278$, $w = 74$, and $h = 271$. Now click the OK button. A button of that size was used for a Go and a Done button on the Tester Interface.

To change the title on the button from Button to Go, you may either select the submenu Attributes from the Button Inspector and enter a new name or double click on the button until the title is highlighted and then type in a new title. By selecting the Font Panel in the Format submenu from the main IB menu, you may choose a desired font size for your title—in this case, Helvetica, bold, 24.0 points. In the Attributes submenu, you can also alter the button's borders and the alignment of titles, as well as how the button responds when pushed (whether it acts as an on/off button or momentarily changes to display an alternate title, etc.) But for now, let the button take its default qualities (e.g., bordered with central alignment of its title and its type Mom. Push).

To create a Done button equal in size to the Go button, do one of two things. The first is to grab another button from Palettes and repeat the previous procedure of dragsizing or numerical manipulation. A second faster option is to select the Go button and, utilizing the IB Edit submenu, select Copy and then select Paste. This will create an identical button for you to place where you wish. Again using the Button Inspector's Miscellaneous function, place the new button at the same y coordinate as the first button (278) and at the new x coordinate of 621. Change the title to Done.

Periodically you should save your work. Within the top bar at the right of the File window (UNTITLED—/user name) is a small Saved indicator that informs you that the file has been saved (a solid ×) or that it contains unsaved changes (a broken ×). See and compare the top right × of Figure 2b (saved) and Figure A1 (unsaved). At the moment, your file has not been saved, so the small square at the top right of the black bar contains a broken ×. When you eventually save it, with the Save command found under the IB submenu File, the × will be solid. The interface file can be identified by a .nib file extension. The File window (the UNTITLED—/user name window; see Figure A1) will take the name that you saved your application under, followed by your user name. If you saved the file under a name other than UNTITLED, the File window and your small application menu will accommodate the new title.

Creating sliders. To create the row of sliders, we will fashion a prototype slider and copy it to produce a row of eight. First drag over a vertical slider from the Palettes window and place it anywhere. (If the Palettes window is partially hidden, click anywhere on it and the Palettes window will become the foreground.) Note that the control points appear on the sliders as well. Unlike buttons, which can be manipulated in height and width, the slider can only change in its height. The width is always 16 points.

Select the vertical slider and the Miscellaneous submenu in the Slider Inspector to change its position within the window by changing the x and y values. In the Auditory Sequence Tracker, the slider height selected was 361 points and the default width was 16 points. Set the first slider at the coordinates $x = 210$, $y = 216$, and $h = 361$.

Viewing the Attributes of the slider in Inspector, note that the default maximum, minimum, and current values set the slider button square in the middle. The experiment, however, requires that the slider buttons start at the bottom. Set the minimum and current values to 0 and click OK; the slider button will drop to the bottom.

Click on the slider so that its control points are visible, and use the IB edit functions Copy and Paste, found under Edit, to create another slider. With the Miscellaneous x and y values, set the second slider at $x = 250$, $y = 216$. Using the *second* slider, copy and paste for a total of eight sliders. Now we will use an alignment command to make all the sliders an equal distance apart, given the dimensions provided by the first two sliders. Click anywhere on the inside of the main window (except on an object) and drag the cursor. A marquee will appear; drag it so that it encircles part of all the sliders. This selects the control points of all sliders. Now, select from the IB submenu Layout, the submenu Align and its submenu command Make Row. (Hint: the marquee will select the control points of any object it touches; it need not surround the entire object.) At this point, your window should have eight sliders in a row and two large buttons on either side.

Adding numbers and reference lines. To provide numerical reference points for the sliders, we will produce rows and columns of numbers by creating a prototype object; from that matrix, set the spacing and change the matrix of 8s to the appropriate numbers. This method may seem a little tedious, but it is the most effective. Through the creation of an 8 × 8 matrix, the numbers can be treated as a unit (size, position, and other characteristics for all can be changed simultaneously), and in the long run this is more efficient.

First, grab the Title object from the Palettes window and drag it over to the main window. Double click the object until the title is highlighted. Change the title to the number 8, with the font at Helvetica, size 14.0 points. To create the matrix, first reduce the surrounding area by closing in the control points. This elimination of the extra space surrounding the 8 will result in a more accurate matrix. In Attributes, change the characteristics of the 8 so that its text is white, and the border and background invisible (i.e., both the Background Gray and Border option at the far left). Be sure to press OK to activate the commands. Utilizing the Miscellaneous function, place the object at $x = 231$ and $y = 547$. Its position should be between the first two sliders. To create a matrix of 8s, Alternate-drag (that is, *simultaneously* press the Alternate key on the keyboard and click on a control point, which will turn white, and drag) the right bottom control point to create an evenly spaced 8 × 8 matrix of 8s (if the present action only resulted in enlarging the space surrounding the 8, try again making sure that the Alternate key is pressed *first* before you click on the control point and that the Alternate key and mouse button remain depressed while you drag). To widen the space between the numbers, Command-drag (using the Command key). Use the Command-drag function to space the matrix so that one row of numbers is between adjacent sliders and the columns are almost as long

as the slider (see Figure 1). Note that the number matrix now overlaps the sliders. To solve this problem, use the Attributes menu to ensure that the background of the matrix is invisible. Use the Send to Back command under Layout to send the matrix behind the sliders so that the overlap (which is still there, only the background is invisible) does not interfere with the functioning of the sliders in the test mode. Select the Cells Match Prototype option (in Matrix Inspector) and click OK. Double click on each of the 8s and change accordingly, so that the end result is ascending rows of numbers as in Figure 1.

Finally, reference lines are needed to assist the subject in determining the appropriate placement of the sliders. These lines may be created by utilizing keyboard characters, a Title object, and the Alternate-drag and Command-drag functions. As with the previous instructions, drag a Title object from Palettes, and double click until "Title" is highlighted and ready to be renamed. Using the Alternate-dash character (–, produced by simultaneously pressing the modifier key Alternate and the dash symbol found in the number row; the dash alone will only produce a dashed line as opposed to a solid line), create a solid line long enough to intersect the eight sliders and then some. Move this new line so that it bisects the row of 8s. Alternate-drag to create a total of eight lines, and then Command-drag so that their placement will bisect their corresponding numbers; now make the background of the matrix invisible and "Send to Back" using the command found under Layout.

Numbering sliders. To number the sliders, drag the predefined object Title from the Palettes window and place it under the first slider. Double click the object until the title is highlighted. Change the title to the number 1 and close in the control points to reduce its surrounding area; place at $x = 210$, $y = 190$. Alternate-drag to create eight 1s and then Command-drag to space them across the width of the sliders. (If you have not first condensed the space occupied by the Title, you will note that when you Alternate- and Command-drag the 1s they may not line up with the sliders.) Double click on each of the last seven 1s, and in an ascending numerical order change the numerical titles.

Labelling *x*- and *y*-axes. Drag another Title object onto the main window, double click until highlighted, and type in the title Order of Presentation of Tones. Place the title at the bottom center of the eight sliders, $x = 223$, $y = 157$. To create the High and Low titles on the *y*-axis, drag two more Title objects and type in the appropriate words. Copy and paste both titles, and place them at each corner of the slider group—Low left and right bottom, High left and right top.

Feedback fields. In order for the subjects to receive feedback, a predefined Field object is required. Move this object from the Palettes window to the bottom right of the sliders. To change the labels, double click on the word Field1 to highlight the label and its accompanying text field. Double click again to highlight only Field1, and type in Trial Number. Do the same for Field2, and change the label to Number Wrong. Using Miscellaneous, place the fields at the bottom right of the sliders ($x = 570$, $y = 60$).

Boxed instructions. To create the instructions box at the bottom of the display, place two predefined Text objects at the bottom center and size one to fit your instructions and the other to fit the title Instructions (the separate boxes are required because a text field will recognize only one font). The text object will automatically adjust its size to fit the text. However, it is more likely to wrap the words around than to adjust the width. Simply alter the width of the text object after the typing is complete, and the text will adjust itself to the new width. (See Figure 1 for the set of instructions.) Again, save your work.

Testing the Interface

Now all graphical aspects of the interface are complete. With the Test Interface command under the submenu File, the response of the application can be tested at this preliminary stage. Since this is only the skeleton without programming, the buttons will not start the program or play a tone sequence. They will, however, respond as they would in the experiment when activated with the mouse. Thus, clicking on the Go button highlights it, and clicking on the slider knob and moving the cursor will move the slider button. (If your sliders do not cooperate, check to see if the number matrix was sent to the back of the sliders.) To quit the testing mode and return to IB, click the Quit command on the menu.

Adapting the Interface for a Different Task

In this section, we describe how to alter the interface with tone patterns for adults (see Figure 1) to one with animal sounds for children (see Figure 3).

Preliminary Changes

Before beginning to make changes, save your IB file under a different name so that new changes will not destroy your saved file. Select Save As under File and give it a new name.

To make the adaptation, some sliders and titles must first be eliminated from the original interface. By clicking on the window and dragging a marquee around the last four sliders and touching the area encompassing the text fields as well as titles in that area, the selected items can be easily removed. You will note that the marquee can cover a large area and select a number of objects. Since the line and number matrices were under the selected sliders, these will be deleted as well. Now, to remove the selected objects, press the delete key or use the command Cut in the Edit submenu to remove these items. On the other side, remove the High and Low titles.

To accommodate the larger application, increase the window size to $w = 869$ and $h = 666$. Change the instructions to match Figure 3. Alter the size of the Go and Done button (using Miscellaneous) to a height of 269 points and a width of 131. The Done button is placed at $x = 696$, $y = 255$ and the Go at $x = 58$, $y = 255$. Double click the Done button to change its name to Stop.

Set the sliders so that they are 106 points apart and increase their size to 206. Add a new Title object, Alternate-drag to create four titles, and double click to change them to 1st, 2nd, and so on.

Adding Animal Icons

Next we need animal pictures, which can be borrowed from the Digital Webster provided by the NeXT. In File Viewer, select the NextApps and double click to start the Webster application. If you ask for the definition of dog, another window should appear supplying the picture of a dog. (If the image of the dog does not appear, first view the options under Preferences ... to see if pictures are selected to be viewed; then check for a picture directory under NextLibrary/References/Webster-Dictionary directories to see if your copy of the Digital Webster is equipped to provide pictures.) To utilize the image provided by the NeXT, we need to screen grab the image. Screen grabbing is done through the application Icon.app. The following instructions will

inform as to how an image is grabbed, scaled down to fit, and saved for use in the new application.

Icons. Icons are small pictorial or graphic representations of an application, file, directory, or document. They are created by using Icon: The Pixel Manipulation Device, a demo application available on the NeXT. This program also enables one to create designs, logos, images, or pictures for use in the Interface Builder or other applicable programs. To build an icon or in this case manipulate an image already provided, launch the Icon Builder from the File Viewer's NeXTDeveloper/Demos directory. Double click on the application name Icon.app. Once the application is launched, the main menu entitled Icon appears along with Tools and the Inspector Panel. Select New under the Image submenu. A blank window appears, called Untitled1.tiff; this window is your canvas. The Tools palette contains the different drawing tools you may use. The Inspection panel is similar to the IB Inspector in that its options change with tool selection. You may want to first experiment with this application to become familiar with its capabilities.

Screen grabbing. The Icon application also allows for screen grabbing an image and manipulating its dimensions, in this case Digital Webster's image of a dog. Under Image select Grab . . . , and the cursor will turn into a small right angle. Move the right angle to the dog window and drag a marquee around the dog. When the mouse button is released, the grabbed image will appear within an Untitled2.tiff window.

Reducing the image. Note that there are pointer lines and numbers on the picture. Use your ingenuity and your newly found Icon skills to eliminate this irrelevant material if you wish, and make the image resemble that in Figure 3. For the purpose of this exercise, it is not necessary to remove the irrelevant material. We must, however, reduce the size of the dog so that it will fit on buttons in the new Auditory Sequence Tracker. Drag a marquee around the dog (by clicking on the bottom right square in Tools, the marquee is activated) or click on Select All under Edit, which will automatically put a marquee around the entire contents of the window. The Inspector Panel will switch its display to Marquee/Lasso Options. Select the scale effect, which is the third button from the bottom right. When that button is selected, the cursor will change to a crosshatch of arrows. Click and hold down the mouse button inside the marquee, then drag in the direction in which the image is to be scaled. When the mouse is let up or unclicked, the marquee is reimaged at its new scale and the scale button is no longer highlighted, thus ending the operation. In order to equate the size of the animals, we need a set of coordinate reference points. The coordinates display is found on the Tools panel. When the coordinates display is activated or selected (by clicking on the button entitled Coordinates) the first two numbers in the display will be the x and y scale values, which are updated as you scale the marquee; the second set will be the scaling percentage. To proportionally scale the image, make sure these two x,y values are equal at the end of the scale operation. For present purposes, scale all animals to the x and y coordinates of 90 and 90, respectively. Clicking outside of the scaled marquee will cause it to vanish; you will then have to repeat the process.

Saving. If this new reduced and marqueed image is to your liking, save it by clicking Save under the submenu Image. A Save Image panel will open. From here, you can save the image within the marquee instead of the entire window by clicking the SAVE DOCUMENT button (with the image of an island and palm tree) so that only the palm tree is marqueed and the button now reads SAVE SELECTION. Type in the title of your new icon, save it in your desired file or location, and OK (a .tiff file extension will be added). Only the reduced image (or icon) is saved. The window will, however, still be called Untitled.tiff, and the Save indicator at the top right shows that your window has not been saved. This is true; only your selected section of the window has been saved. To double-check that only the icon has been saved, use the submenu command Image and Open to open your saved section and assure yourself of its existence. Using the Digital Webster and Icon, create and reduce .tiff files for the remaining animals.

Adding .TIFF Files to Interface Builder

In the Interface Builder, you must add the new animal icon to the icon suitcase in order to utilize it. From the File Viewer select and drag each .tiff file icon (TIFF) from the File Viewer to the icon suitcase in your Interface Builder file. The case will open to store the icon, and a small window will appear to show your stored icons along with five standard ones. To place the icons beside the slider, buttons are needed. A button need not be sized to accept an icon; it will adjust itself to fit the icon. Icons can be placed on buttons by dragging the .tiff files from the Store window of the suitcase directly to a button. A black outline will surround the button that is immediately below the image to aid in placement. To eliminate the border around the button so that only the icon is visible, select the option under Attributes to remove the borders. Also under Options, turn the Disable switch on, so that if the button is pressed during the running of the program nothing will happen. Follow this procedure for the four animal icons. Proceed with the adaptation of the application. Now that the skeleton has been created, we need only program the application so that when Go is clicked, a sequence of animal sounds will be produced.

(Manuscript received September 24, 1990;
revision accepted for publication June 10, 1991.)