Sound Kit is a system for manipulating and editing digitally sampled sound. It was developed at Apple Computer, Inc. as a research tool for programmable user interfaces known as Kita. The resulting system provides both a tool for this research (both in general and as it applies to music) and a versatile sound manipulation system.

Sound Kit is more than simply an editor for sampled sound. While the functions of splitting, looping, filtering, etc. are available, the real power of the system lies in its ability to be programmed by the user. By this we do not mean programming in the traditional sense. The user does not write algorithms but rather builds new operations from existing ones in ways beyond simple sequencing. In order to have this capability be powerful, Sound Kit deals with the concepts of objects and instantiation (model sounds), generality (extent or range) of operation, graphical presentation of data and operation, user extensibility, and operation modifiability (alteration of operation).

This paper discusses three aspects of Sound Kit. First, the conceptual ideas of such a system as they relate to sound are presented. Then the current interface and operation is described including examples of the system in operation. Finally, some of the implementation that relates to sound systems in general and how those systems interact with other systems (operating systems, composing systems, graphit art systems, etc.) are discussed.

1. WHY A SOUND KIT

1.1. Desiderata for an Editor of Sound

Most editors of sound view the editing process the same way that text editors do. The data is represented by a collection of files, each of which is a linear sequence of some basic unit, a sample. A set of commands is provided for moving around the data and for manipulating it, such as cut, copy, paste, filter, attenuate, re-sample, etc. I feel that this view of the editing process is insufficient, both the sound data model and the command structure are too simple and do not reflect what users of such systems want to achieve.

When editing sound one wants to grasp sound data in ways more relevant to the editing process. Individual sounds want to be grouped and related in the manner that they will be worked with. Operations such as "increase the volume of the chordal answer to each of the tenor's questions" are not possible unless the categorization of which sounds are the tenor's questions and the relationship of answers are expressible to the editor. Some editors add the ability to name points and/or phrases in the sound. This gives at best a syntactic way to organize the sound. While one can name one phrase "Violin C" and another "Violin C sharp" there is no real connection between the two phrases as far as the editor is concerned. While the user may want to manipulate all violin tones, or all C tones, for example, she must rely on her own syntactic conventions for naming phrases to make all she all possible.

Commands are merely syntax for some operation. When editing one wants to perform various operations on the sound, yet one does so indirectly through commands that invoke operations. When working with more complex editing tasks than offered by the command set, editing involves sequences of commands that are in no way related as far as the editor is concerned. There is no way to express the larger operation that is taking place, only the sequence of commands. Some editors allow one to build macros. But these are only a sequences of commands that may be re-played via shorter commands. Macros are simply syntactic substitution of one sequence of commands for another command and do not possess the full power of building new operations from existing ones.

What is needed is a better model for sounds, and a way to work with the operations involved, not just the commands.

1.2. Definition of Kit

"Kit... to personal equipment, esp. as packed for travel to a set of tools or implements & equipment for some particular activity, sport, etc." [Webster's New World Dictionary, p. 778, New York: New World Dictionary/Simon & Schuster].

Kit can refer to any of several things with respect to computer systems. The term kit is used here to mean any computer application that gives to end users the ability to perform tasks previously delegated to programmers. For example spreadsheet programs are kits because they put the ability to program reports in the hands of the people who have to use them.

Figure 1.

ICMC '85 Proceedings
Several properties are naturally common to kits. Building blocks are often used to describe the domain of the application. These building blocks may then be combined in hierarchical or recursive schemes. In essence, the same way one builds programs out of fundamental computing structures, users tell a kit to perform functions in the domain of the application.

The Sound Kit is a kit whose building blocks are sounds, their relationships, and the operations to be performed upon them. To answer the needs stated above, given a good set of building blocks, the Sound Kit should have both a better model of sounds and a way to manipulate operations.

2. THE CURRENT KIT

2.1. First Appearances

The Sound Kit is currently implemented in the Smalltalk-80 programming language on both the Apple Macintosh™ and Macintosh XL™. It is, like most applications in Smalltalk, integrated into the Smalltalk environment and shares much with it. Much of the basic user interface comes from the Smalltalk system: pop-up menus, mouse and cursor meanings, window managers, and so forth. These features are similar to many other window systems. The particulars are not of major importance to the kit and so will not be described here.

Two types of window are used in the Sound Kit: sound views and operation views. The sound views offer a window onto the sounds as waveforms. These views are scrollable and zoomable. The user may add the sound with comments bound in the main associated with each window (figure 1). Within the rest of the Kit these views can be used as a simple sound editor with most of the features of other sound editors (although macros are not offered).

The other type of window, the operation view, adds the features of kits. This view (of which there is only one, which is usually delegated to the right side of the screen) displays the writing operations in graphical form. Figure 2 shows a screen with an operation view on the right after a user has copied a piece of sound and pasted it back twice, thus tripling the sound. Note the iconic arguments that the system has given the operations: a sound and one or two specific times.

2.2. The Model

The main screen of the Sound Kit is based on three basic building blocks: Sounds, Relationships, and Operations.

There are several types of sounds. There are sampled sounds that come from the outside world. There are periodic sounds that are repetitions of other sounds. There are computed sounds that are computed from mathematical functions. There are composite sounds that are sequences of other sounds. The user never needs to worry about what type of sound he is working with; they all respond to the same set of operations (although some have a few extra properties that may be worked with).

Sounds may have relationships among them. A relationship may be considered a relationship of a sound, or function from one sound to another (much like a Plotting function). For example, a relationship of pitch can be specified between a sound and some assigned pitch, or a relationship of derived form could create a modified sound to the sound from which it was derived. Most relationships are asserted by the user as needed, though most relationships are not complete, for example, not all sounds need be related to a pitch. The Sound Kit maintains some relationships automatically; the relationship of 'fifth' between each sound and what kind of sound it is and the relationship of 'fifteenth' among sounds inverted by time.

Lastly, there are operations. Operations are the editing actions that get performed in the system, not to be confused with whatever commands may be used to invoke them. Operations have been divided into three categories: substitutions, performers, and constructions. A substitution is an operation that is used to select one or more objects to be the subject of further operations. A performer is an operation that actually manipulates and changes things such as filtering or copying. A constructor is used to generate operations.

To clarify this, refer to figure 5, below. The is the operation of tripling from above generalized to occur over a whole range of sounds. First a constructor specifies that the rest of the operation should take place on all sounds that have a particular relationship with some fixed string, for example: all sounds whose instrument is piano (the icon is for a relationship and a fixed string). Then a performer extracts the current selection area sound. Several substitutions determine where to copy from and paste to the whole selection and the end of the selection respectively. Lastly, the performer of copying and pasting takes effect.

Other editors do not have substitutions or constructors. Therefore the user is always burdened with the tedium of argument selection and repetitive execution.

2.3. Functionality

Any time a command issued in any of the sound windows, a symbolic representation of the operation performed is added to the operation window. The operation window serves as a history of what has happened. At any time, the user may elect to use an existing operation in the operation window instead of choosing a command from a menu. Thus, in addition to the standard undo operation that is provided, the user can perform a redo of any operation. The usefulness of undo, redo, and a history of operations is well-assisted by the modular inclusion of many existing systems such as Interleaf and the Vajac C-Build. An important difference however, is that here, the history does not record a syntactic textual version of what was executed, but instead the operations themselves in a semantic form that can be manipulated graphically.

When working with the operation window, the user can create new operations by manipulating previous ones. Each operation has a set of one or more argument tabs that may be attached to any object on the screen so as to direct the action of the operation to involve some other data. At that point the operation may be re-executed upon the new arguments. Various operations may be selected at once for re-execution and when this is done, a new operation is formed that consists of their combination.

Figure 2.

ICMC '85 Proceedings
Some other useful Smalltalk services are: spelling correction of typed commands and expressions; null-multiplication with floating point and fractional calculations; text editors; multi-tasking capability.

3. KIT INTERNALS

3.1. Breakdown

It is not within the scope of the paper to go into the details of the internals of the Sound Kit. However, its major components and some of its sub-systems are interesting and are worthy of mention.

Internally the Sound Kit is made up of three parts: the Sound Model, the Operation System, and the Iteration Engine. The Sound Model is responsible for displaying the sounds and doing the work of the operations. The Operation System manages the operations and directs the general analysis. The Iteration Engine keeps track of the relationships that the user creates and invokes as well as performing important parts of the general analysis.

2.4. Environment

Since the Sound Kit is totally embedded in the Smalltalk-80™ standard environment, users may take advantage of the many tools and services that Smalltalk provides. Figure 6 shows a Project, a FileList and a Workspace active within the Editor.

Projects offer a way of managing various different editing tasks. Switching from project to project allows one to completely switch from one set of sounds to another. File Lists offer a flexible view of the file system, both local and across the network. From a list of files, the user can import any sound file into the Sound Kit and begin to work with it. Workspaces provide scratch space for calculations and other manipulations of the Smalltalk environment.

Adding new functionality to the Sound Kit is surprisingly simple. If a new operation is desired for a certain type of sound, for example amplitude normalization of a sampled sound, only the code need be written for it. The rest of the task of adding a function to the editor is automatic. The function is added to a menu. The function is only made available in appropriate circumstances (amplitude normalization wouldn't become a function of silence, for instance). Undoing is provided for. Selection of the new function by the user
automatically makes a new operation that can be manipulated like all others. All this is done by having the system ask each sound, as it is being echoed, what functions are possible. The rest of the system is written to work with the answered list of available functions. Hence, in the multi-tasking environment of Smalltalk, functions can be added while editing is taking place. This feature is very useful while working on the Sound Kit. It is also an asset to more advanced users for they can add new signal processing functions without having to know anything about the Sound Kit's internals or having to touch the Sound Kit's code.

![Figure 6](image)

The Operation System is completely unaware of what it is operating upon. It merely knows proxies for operations (function names) and references to the args lists. Therefore, it is easily extended to other domains. The result of this will be that the Sound Kit can expand to incorporate other things—notes lists are planned for the near future. Hence, a user will not have to be constrained to working in a sound editor for sounds, and a new list editor for note lists, etc. But instead, she can work in one environment, where not only is the method of interaction and operation the same for all domains, but furthermore, interaction between the domains is totally permissible. Thus, operations such as filter a waveform table and raise the pitch of all of all notes that use that table could be easily constructed.

4. CONCLUSION

The Sound Kit is an editor that was built to address editing not as an unconnected sequence of commands in data streams, but as a manipulation of operations on data with semantic structures. Its implementation was done with an eye to extensibility to other domains and for quick prototyping.

The Sound Kit is part of an exploration of new ideas about programming and interface at Apple Computer, Inc. Sound and Music are two of the domains being used as test beds for these ideas. I would like to thank Dan Ingalls for the all the ideas he gave to this project and the many hours spent discussing kits.

Smalltalk-60 is a trademark of the Xerox Corporation.
Macintosh and Macintosh II are trademarks of Apple Computer, Inc.