miniAudicle and ChucK Shell: New Interfaces for ChucK Development and Performance

Spencer Salazar, Ge Wang, and Perry Cook[†]

Department of Computer Science [†](also Music), Princeton University
{ssalazar, gewang, prc}@cs.princeton.edu

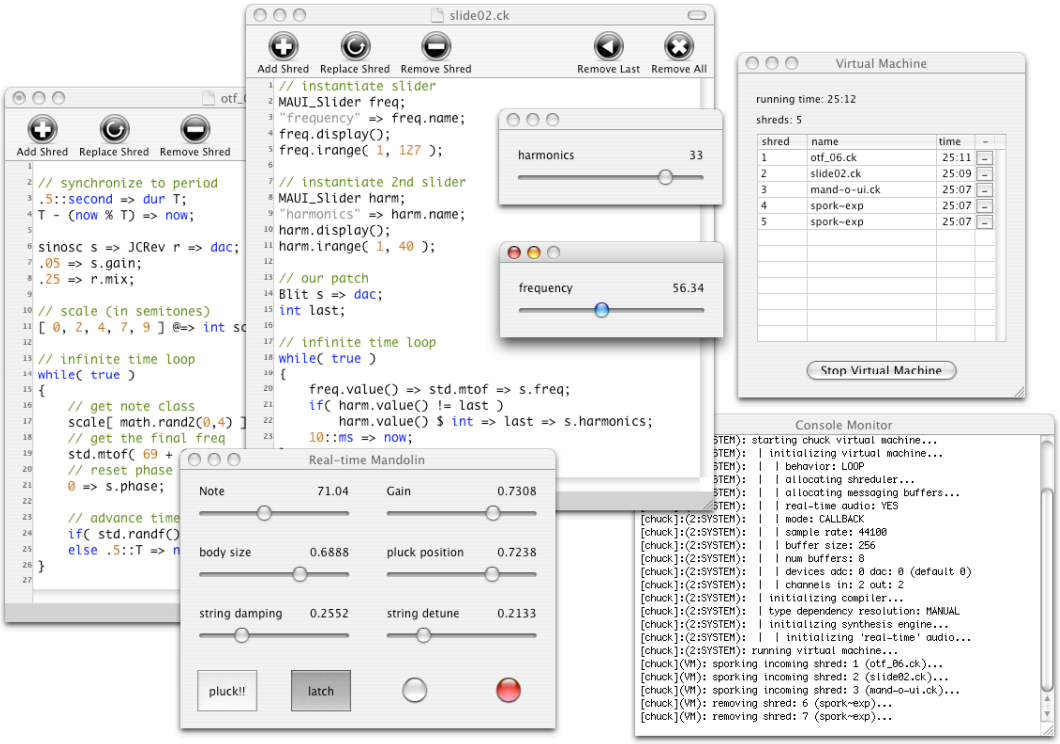

Figure 1. The miniAudicle in action.

Abstract

ChucK, a powerful audio synthesis programming language, currently supporting only a simple command line interface. Accompanying the ongoing development of the ChucK language is the production of two new interfaces for ChucK, the ChucK shell and the miniAudicle. The ChucK shell provides a lightweight method of access to ChucK in a shell-like console environment. The miniAudicle offers a powerful integrated solution to the ChucK development process and a framework for further enhancements to the ChucK programming environment. The miniAudicle also provides a set of generic user interface elements with which programmers can quickly construct graphical interfaces in ChucK programs. The miniAudicle streamlines and simplifies ChucK development, allowing ChucK programmers to focus on design and artistic issues while also exposing ChucK to computer musicians who are unfamiliar with or averse to the command line.

1 Introduction

ChucK, a programming language for audio synthesis (Wang and Cook 2003), affords its programmers a rich digital audio development environment for both traditional write-compile-run style programming and Wang and Cook's on-the-fly programming model (2004). On-the-fly programming is a concept in which programs can be modified and controlled at runtime both precisely and directly, allowing greater power of expression to the programmer. On-the-fly programming typically requires some degree of runtime support in a given language. ChucK provides such capabilities in the form of a virtual machine, which compiles, executes, and associates different ChucK programs as programmed. However, ChucK itself lacks a convenient user interface for its users; to execute ChucK source files, users must familiarize themselves with ChucK's command line interface, accessible only through a shell environment such as `bash`. The Audicle, a graphical

system for monitoring and modifying ChucK programs, presents one solution to this problem, but is still in the early stages of development (Wang and Cook 2004). The purpose of the ChucK Shell and miniAudicle are to provide sophisticated, accessible, and convenient interfaces to the ChucK language and on-the-fly programming.

In this aim, ChucK shell presents users with a native shell-like interface for interaction with the ChucK virtual machine. miniAudicle is a graphical user interface for developing, running, and modifying ChucK programs, providing a framework for extending the expressive capabilities of ChucK in possibly unforeseen ways, and making ChucK more accessible to individuals with little or no experience with command lines or software programming.

In Section 2, we shall discuss related work. In Section 3, we shall present the design and implementation of the ChucK shell, and section 4 describes the design and implementation of miniAudicle. Section 5 discusses more broadly the successes and shortcomings of ChucK shell and miniAudicle, plans for future work, and then concludes.

2 Related Work

One existing extension of the traditional ChucK interface is the Audicle, an advanced 3 dimensional graphical interface for interacting with the ChucK environment (Wang and Cook 2004). The Audicle is a multi-platform, Open-GL based application supporting editing, execution, and modification of ChucK programs, as well as advanced monitoring of ChucK virtual machine state. While having a well-developed feature set, the Audicle has several limitations. The Audicle uses OpenGL, a 3D graphics programming API, to generate its user interface. Though the Audicle's interface is highly functional and visually impressive, by requiring OpenGL programming, the complexity and time required to implement new functionality in the Audicle is increased. Additionally, by implementing its own user interface primitives, the Audicle intentionally differentiates its interface from the native graphical user interface API of its host operating system. This augments the barrier to accessibility, requiring users to learn a different interface structure to effectively use the Audicle.

Other software environments for digital audio synthesis include PureData (Puckette 1997), a visual audio programming language, SuperCollider (McCartney 1996), and CSound (Vercoe 1990).

3 ChucK Shell

The goals of the ChucK shell are to simplify and streamline basic tasks in the ChucK environment, and potentially provide facilities for more advanced tasks as needed. The ChucK shell is accessed using the chuck command line application, by specifying the `--shell` command line option. This command alone will invoke the ChucK shell and also create a ChucK virtual machine, in separate threads; the shell is implicitly connected to this virtual machine. It is possible to suppress creation of the virtual machine by specifying the `--empty` command, spawning a shell process which must then be explicitly connected to a different virtual machine.

The ChucK shell has been designed using fully reusable and modular components, allowing it to be easily embedded in both the miniAudicle and the Audicle.

3.1 Shell Commands

The basic ChucK shell commands are `add`, `remove`, `replace`, `removeall`, and `removelast`. These commands can be invoked by name, or by an associated symbolic name (see Figure 2). These commands map directly to the ChucK on-the-fly programming commands of the same name. Additionally, using the `vm @` command, users can specify a network address and port number to which these commands are sent (ChucK virtual machines typically bind to a port on the host machine to listen for commands over the network). ChucK shell also replicates selected standard shell commands such as `ls`, `cd`, `pwd`, `source`, and `alias`.

```
chuck %> + sequencer.ck
chuck %> = 1 sequencer-new.ck
chuck %> - 1
chuck %> removeall
```

Figure 2. Example ChucK shell commands.

3.2 Inline Coding

The inline coding functionality of ChucK shell allows users to quickly execute small snippets of ChucK code from within the shell, without having to employ an external source file editor. Small snippets of code or even entire programs can be written without having to leave the shell, although the latter is not necessarily advisable. This functionality is primarily intended for the context of a performance, in which the small time to open a new file in an editor is significant, and it is only necessary to run some small snippet of code that does not already exist as a file on disk.

```
chuck %> {
code  1>   sinosc s => dac;
code  1>   while( true ) {
code  2>     1::second => now;
code  2>   }
code  1> }
chuck %>
```

Figure 3. Example of inline coding. The lexical scope level is printed in the command prompt.

Inline code is preceded by an open bracket "{" and terminated by a matching close bracket "}" (see Figure 3). ChucK shell tracks open and close brackets so that intermediate close brackets are not misinterpreted as the end of the code fragment. Code fragments can be saved to disk after they are sent to the virtual machine.

4 miniAudicle

The miniAudicle offers an integrated, graphical environment for developing audio software using the ChucK programming language. miniAudicle features a text editor, an embedded virtual machine, a virtual machine monitor, a stdin/stderr monitor for displaying log and error messages from the virtual machine, an embedded ChucK shell, and support for on-the-fly programming commands like add, remove and replace. miniAudicle also supports creation and usage of typical graphical user interface widgets directly from ChucK code, chiefly for modifying program behavior and parameters at runtime in a well-defined but expressive manner. Currently, the miniAudicle uses the Cocoa API in the Mac OS X operating system to render its graphical user interface.

The basic layout of the miniAudicle comprises 4 windows. The document window encapsulates ChucK source code files and presents them for editing, with common editing features such as syntax highlighting. Several buttons in the document window correspond to on-the-fly programming commands, and where applicable will add, remove, or replace the file open in the editing window. Multiple document windows can be created, either with a new blank document or an existing document from disk, as is typical behavior with text editors.

The second window is the virtual machine monitor (see Figure 4). The virtual machine monitor can be used to start and stop the virtual machine. The virtual machine runs in the address space of the miniAudicle. This virtual machine accepts on-the-fly programming commands from both the command line ChucK client and the ChucK shell, by binding to port 8888 on the host machine. When the virtual machine is on, the virtual machine monitor also displays a table of all currently executing ChucK programs on the virtual machine, showing the unique program identification number, the filename of the program, and the running time of that program. Additionally, a small button in each entry of the table can be pressed to remove the ChucK program associated with that entry. The virtual machine monitor also prints the running time of the virtual machine and the number of programs currently running.

The third window is the console monitor (see Figure 5). The console monitor simply redirects the stdin and stderr files to itself, and displays them on screen. In this way, log and error messages printed by the virtual machine to stdout and stderr can be displayed to the user without requiring an actual terminal window open.

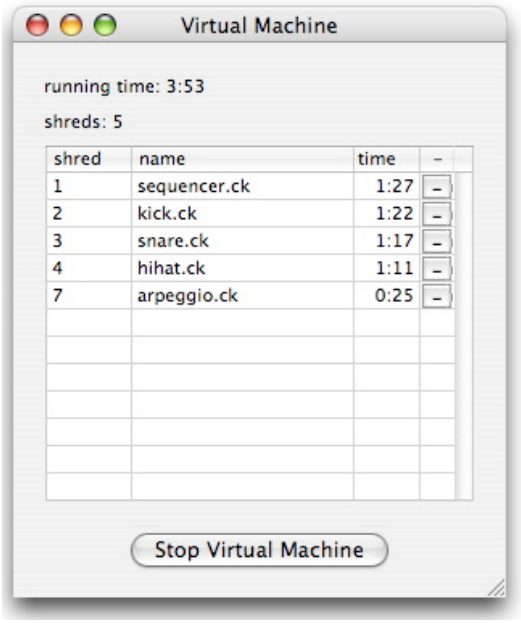

Figure 4. The miniAudicle virtual machine monitor. A "shred" refers to a running ChucK process.

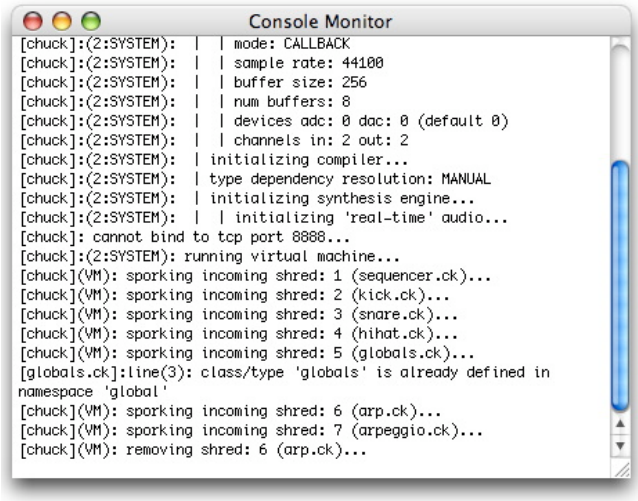

Figure 5. The miniAudicle console monitor.

The fourth window is the ChucK shell window. This window is essentially a simple terminal emulator that directs its input to a ChucK shell instance, and prints any output from the ChucK shell.

Additionally, the miniAudicle affords ChucK programmers the capability to dynamically generate new user interfaces. The miniAudicle user interface elements, or MAUI Elements, are a set of common user interface widgets that can be created, manipulated, and queried within a ChucK program. miniAudicle currently provides several such elements, including sliders (Figure 6) and buttons for user input, and simulated light emitting diodes (LED's) for

```
MAUI_Slider slider;
slider.range( 100, 10000 );
slider.value( 100 );
slider.name( "frequency" );
slider.display();

sinosc s => dac;

while( true )
{
    slider.value() => s.freq;
    slider => now;
}
```

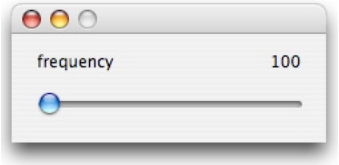

Figure 6. Example ChucK code (top) for invoking the MAUI Slider element. This example generates a sine wave. It also displays a slider (bottom), with values between 100 and 10000. The frequency of the sine wave is mapped to the value of the slider.

providing feedback to the user. An additional "view" element allows both logical groupings of elements and physical grouping and positioning of elements within a window (Figure 7). Elements not placed in a preexisting window will automatically produce a window in which to place themselves.

MAUI Elements exist in ChucK as built-in classes that can be instantiated and accessed as any other ChucK class. Each element's associated class has member functions for creation, accessing and setting properties, display, and destruction. Additionally, using ChucK's built-in event notification support, ChucK programs can elect to block execution until a user has modified the external state of an element (for example, Figure 6 top). Notably, ChucK programs utilizing MAUI Elements are unable to run outside of the miniAudicle, since support for MAUI Elements must be explicitly included in the miniAudicle's ChucK virtual machine implementation.

5 Conclusions and Future Work

The ChucK shell improves upon ChucK's basic interface by reducing the keystrokes required for basic tasks and providing additional functionality like inline coding. The miniAudicle offers an integrated environment for ChucK development, using native GUI APIs of Mac OS X. Additionally, the miniAudicle supplies an extensible framework for accessing graphical user interface widgets from ChucK programs.

Future development plans for miniAudicle include porting to Microsoft Windows and Linux systems, automatic completion of code during editing, and implementation of additional MAUI Elements. Additionally, plans exist to embed portions of the Audicle within the miniAudicle.

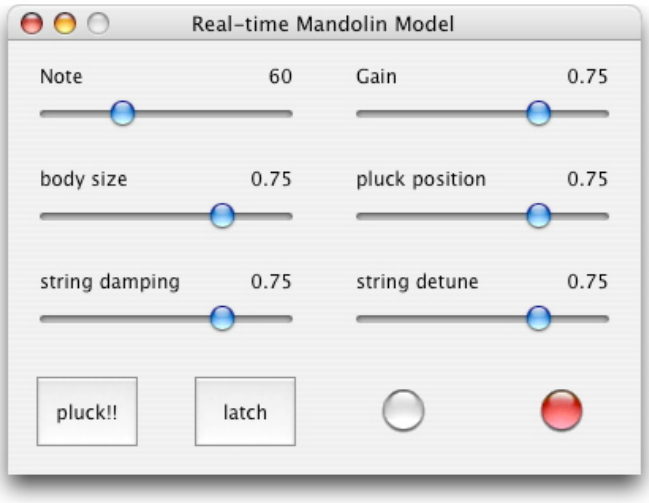

Figure 7. An interface constructed in miniAudicle.

6 Acknowledgements

Thanks to Adam Tindale for his help in documenting the ChucK shell and the chuck-users mailing list for their feedback and suggestions.

References

McCartney, J. 1996. "SuperCollider: A New Real-time Synthesis Language." In Proceedings of the International Computer Music Conference. International Computer Music Association.

Puckette, M. 1997. "Pure Data." In Proceedings of the International Computer Music Conference. International Computer Music Association, pp. 269-272.

Vercoe, B. and D. Ellis. 1990. "Real-Time CSOUND: Software Synthesis with Sensing and Control." In *Proceedings of the International Computer Music Conference*. International Computer Music Association, pp. 209-211.

Wang, G. and P. R. Cook. 2004. The Audicle: A Context-Sensitive, On-the-fly Audio Programming Environ/mentality." In *Proceedings of the International Computer Music Conference*. International Computer Music Association.

Wang, G. and P. R. Cook. 2003. ChucK: a Concurrent and On-the-fly Audio Programming Language. In *Proceedings of the International Computer Music Conference*. International Computer Music Association, pp. 219-226.

Wang, G. and P. R. Cook. 2004. On-the-fly Programming: Using Code as an Expressive Musical Instrument. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. pp. 138-143.