CECILIA AND TCLCSOUND

Jean Piché
Faculté de Musique
Université de Montréal
Montréal, Canada
jean@piche.com

Victor Lazzarini
Music Technology Laboratory
National University of Ireland
Maynooth
Victor.Lazzarini@nuim.ie

ABSTRACT

This article discusses some developments relating to environments for Csound programming, composition and performance. It introduces the Csound 5 API and discusses its use in the development of a TclTk scripting interface, TclCsound. The three components of TclCsound are presented and discussed. A number of applications, from simple transport control of Csound to client-server networking are explained in some detail. The new multi-platform version of CECILIA is presented. Cecilia is the first Csound frontend to use the functionalities of TclCsound.

1. INTRODUCTION

The Csound music programming system [1] is currently the most complete of the text-based audio processing systems in terms of its unit generator collection. Csound hails from a long tradition in Computer Music. Together with cmusic [2], it was one of the first modern C-language-based portable sound compilers [3], when it was released in 1986. Due to its source-code availability, first from an MIT ftp server and then from the DREAM site at Bath, it was adopted by composers and developers world-wide. These brave people developed Csound into a formidable tool for sound synthesis, processing and computer music composition. Its latest version, Csound 5 [4] has close to one thousand opcodes, ranging from the basic table lookup oscillator to spectral signal demixing unit generators.

The usability of command line languages for music creation has always made it difficult to promote their wide-spread use by composers. In the hope of bridging this approachability gap, a graphical and highly interactive front end was developed for the language. Cecilia [5] is built as a tcl/tk program that issues calls to an independent Csound process through Unix pipes. It proposes a graphical interface, sliders, buttons and an editor to make orchestras and scores. Cecilia also includes a simple algorithmic score generator known as Cybil. We have developed a new version of the interface where the functionalities of Cecilia are extended to all platforms

Many important changes have been introduced in Csound5, which involved a complete redesign of the software. This resulted not only in a better software, from an engineering perspective, but in the support for many new possible ways of using and interacting with Csound. An important development has been the availability of a complete C API (the so-called 'Host API', which was, in fact, already partially partially present in earlier versions). The API can be used to instantiate and control Csound from a host application, such as TclCsound/Cecilia.

2. THE CSOUND 5 API

The Csound 5 Host API allows the embedding of the audio processing system under other 'host' software. Effectively, Csound is now a library, `libcsound`, that can provide audio services, such as synthesis and processing, for any application. This allows for complete control over the functioning of the audio engine, including transport control, loading of plugins, inter-application software bus, multi-threading, etc.. A 'classic' Csound command-line program can now be written based only on a few API calls:

```
#include <csound.h>

int main(int argc, char **argv) {

int result;

/* the csound instance */
CSOUND *cs;

/* initialise the library */
csoundInitialize(&argc, &argv, 0);

/* create the csound instance */
cs = csoundCreate(NULL);

/* compile csound code  */
result = csoundCompile(cs, argc, argv);

/* this is the processing loop */
if(result) while(csoundPerformKsmps(cs)==0);

/* destroy the instance */
csoundDestroy(cs);

return 0;
}
```

The Csound API can be used in many applications; the development of frontends is the most obvious of these. A good example of its application is found on the `csoundapi~` Class, which provides a multi-instantiable interface to Csound 5 for Pure Data. The Csound API is the basis for TclCsound [5], a Tcl/Tk extension, discussed in the next section.

3. TCLCSOUND

The classic interface to Csound gives you access to the program via a command-line such as

```
csound -odac hommage.csd
```

This is a simple yet effective way of making sound. However, it does not give you neither flexibility nor interaction. With the

advent of the API, a lot more is possible. At this stage, TclCsound was introduced to provide a simple scripting interface to Csound. Tcl is a simple language that is easy to extend and provide nice facilities such as easy file access and TCP networking. With its Tk component, it can also handle a graphic and event interface. TclCsound provides three 'points of contact' with Tcl:

1. a csound-aware tcl interpreter (`cstclsh`)
2. a csound-aware windowing shell (`cswish`)
3. a csound-commands module for Tcl/Tk (`tclcsound` dynamic lib)

3.1. The Tcl interpreter: cstclsh

With cstclsh, it is possible to have interactive control over a Csound performance. The command starts an interactive shell, which holds an instance of Csound. A number of commands can then be used to control it. For instance, the following command can compile Csound code and load it in memory ready for performance:

```
csCompile -odac hommage.csd -m0
```

Once this is done, performance can be started in two ways: using `csPlay` or `csPerform`. The command

```
csPlay
```

will start the Csound performance in a separate thread and return to the cstclsh prompt. A number of commands can then be used to control Csound. For instance,

```
csPause
```

will pause performance; and

```
csRewind
```

will rewind to the beginning of the note-list. The `csNote`, `csTable` and `csEvent` commands can be used to add Csound score events to the performance, on-the-fly. The `csPerform` command, as opposed to `csPlay`, will not launch a separate thread, but will run Csound in the same thread, returning only when the performance is finished. A variety of other commands exist, providing full control of Csound.

3.2. Cswish: the windowing shell

With Cswish, Tk widgets and commands can be used to provide graphical interface and event handling. As with `cstclsh`, running the `cswish` command also opens an interactive shell. For instance, the following commands can be used to create a transport control panel for Csound:

```
frame .fr
button .fr.play -text play -command csPlay
button .fr.pause -text pause -command csPause
button .fr.rew -text rew -command csRewind
pack .fr .fr.play .fr.pause .fr.rew
```

Similarly, it is possible to bind keys to commands so that the computer keyboard can be used to play Csound.

Particularly useful are the control channel commands that TclCsound provides. For instance, named IO channels can be registered with TclCsound and these can be used with the invalue, out-value opcodes. In addition, the Csound API also provides a complete software bus for audio, control and string channels. It is possible in TclCsound to access control and string bus channels (the audio bus is not implemented, as Tcl is not able to handle such data). With these TclCsound commands, Tk widgets can be easily connected to synthesis parameters.

3.3. A Csound server example

In Tcl, setting up TCP network connections is very simple. With a few lines of code a Csound server can be built. This can accept connections from the local machine or from remote clients. Not only Tcl/Tk clients can send commands to it, but TCP connections can be made from other software, such as, for instance, Pure Data (PD). A Tcl script that can be run under the standard tclsh interpreter is shown below. It uses the TclCsound module, a dynamic library that adds the Csound API commands to Tcl.

```
# load tclcsound.so
#(OSX: tclcsound.dylib, Windows: tclcsound.dll)
load tclcsound.so Tclcsound
set forever 0

# This arranges for commands to be evaluated
proc ChanEval { chan client } {
if {[catch { set rtn [eval [gets  $chan]]} err]}{
puts "Error: $err"
} else {
puts $client $rtn
flush $client
}
}

# this arranges for connections to be made
proc NewChan { chan host port } {
puts "Csound server: connected to $host on port
$port ($chan)"
fileevent $chan readable [list ChanEval $chan $host]
}

# this sets up a server to listen for
# connections
set server [socket -server NewChan 40001]
set sinfo  [fconfigure $server -sockname]
puts "Csound server: ready for connections on
port [lindex $sinfo 2]"
vwait forever
```

With the server running, it is then possible to set up clients to control the Csound server. Such clients can be run from standard Tcl/Tk interpreters, as they do not evaluate the Csound commands themselves. Here is an example of client connections to a Csound server, using Tcl:

```
# connect to server
set sock [socket localhost 40001]

# compile Csound code
puts $sock "csCompile -odac hommage.csd"
flush $sock

# start performance
puts $sock "csPlay"
flush $sock

# stop performance
puts $sock "csStop"
flush $sock
```

As mentioned before, it is possible to set up clients using other software systems, such as PD. Such clients need only to connect

to the server (using a netsend object) and send messages to it. The first item of each message is taken to be a command. Further items can optionally be added to it as arguments to that command.

3.4. TclCsound as a language wrapper

It is possible to use TclCsound at a slightly lower level, as many of the C API functions have been wrapped as Tcl commands. For instance it is possible to create a 'classic' Csound command-line frontend completely written in Tcl. The following script demonstrates this:

```
#!/usr/local/bin/cstclsh
set result 1
csCompileList $argv
while { $result != 0 } {
set result csPerformKsmps
}
```

This script is effectively equivalent to the C program shown in section 2. If saved to, say, a file called `csound.tcl`, and made executable, it is possible to run it as in

```
csound.tcl âĂŞodac hommage.csd
```

With TclCsound, it is possible to transform the popular text editor *emacs* into a Csound scripting/performing environment. When in Tcl mode, the editor allows for Tcl expressions to be evaluated by selection and use of a simple escape sequence (ctrl-C ctrl-X). This facility allows the integrated editing and performance of Csound and Tcl/Tk code.

3.5. TclCsound as a Csound performance environment

In Tcl it is possible to write score and orchestra files that can be saved, compiled and run by the same script, under the *emacs* environment. The following example shows a Tcl script that builds a Csound instrument and then proceeds to run a Csound performance. It creates 10 slightly detuned parallel oscillators, generating sounds similar to those found in Risset's *Inharmonique*.

```
load tclcsound.so Tclcsound

# set up some intermediary files
set orcfile "tcl.orc"
set scofile "tcl.sco"
set orc [open $orcfile w]
set sco [open $scofile w]

# This Tcl procedure builds an instrument
proc MakeIns { no code } {
global orc sco
puts $orc "instr␣$no"
puts $orc $code
puts $orc "endin"
}

# Here is the instrument code
append ins "asum␣init␣0␣\n"
append ins "ifreq␣=␣p5␣\n"
append ins "iamp␣=␣p4␣\n"

for { set i 0 } { $i < 10 } { incr i } {
append ins "a$i␣␣oscili␣iamp,
␣␣␣␣␣␣␣␣ifreq+ifreq*[expr␣$i␣*␣0.002],␣1\n"
}

for { set i 0 } {$i < 10 } { incr i } {
    if { $i } {
        append ins "␣+␣a$i"
    } else {
        append ins "asum␣=␣a$i␣"
    }
}

append ins "\nk1␣linen␣1,␣0.01,␣p3,␣0.1␣\n"
append ins "out␣asum*k1"

# build the instrument and a dummy score
MakeIns 1 $ins
puts $sco  "f0␣10"

close $orc
close $sco

# compile
csCompile $orcfile $scofile -odac -d -m0

# set a wavetable
csTable 1 0 16384 10 1 .5 .25 .2 .17 .15 .12 .1

# send in a sequence of events and perform it
for {set i 0} { $i < 60 } { incr i } {
    csNote 1 [expr $i * 0.1] .5 \
      [expr ($i * 10) + 500] [expr 100 + $i * 10]
}

csPerform

# it is possible to run it interactively as
# well
csNote 1 0 10 1000 200
csPlay
```

The use of such facilities as provided by *emacs* can emulate an environment not unlike the one found under the so-called 'modern synthesis systems', such as SuperCollider (SC). In fact, it is possible to run Csound in a client-server set-up, which is one of the features of SC3. A major advantage is that Csound provides about three or four times the number of unit generators found in that language (as well as providing a lower-level approach to signal processing, in fact these are but a few advantages of Csound).

4. THE NEW CECILIA

Over the years, Cecilia [6] has made the utilization of Csound much more intuitive and productive. Cecilia provides a library of common sound processing modules that are ready to use for the By providing an environment where one can quickly build interfaces from tk widgets linked to Csound parameters, explorations of sound processing algorithms becomes much more convivial and interactive.

4.1. Key Cecilia concepts

Technically, Cecilia consists of time variant functions that are sent to Csound from real-time sliders on screen, from physical sliders such as MIDI fader boxes or from a screen graphing window. These objects use global Csound variables to send their data to the relevant instrument parameters. All data from interface objects is communicated to Csound with i-time stdin console messages. A system of a single tagged instrument per controller allows for coding a large number of simultaneous channels of control data.

All controllers, whether graphical or physical, are assigned a unique instrument number and, at performance time, will receive data exclusively from the Csound process stdin. Every time a gesture is detected on the interface or a physical slider, the following tcl process is called and the current value of the object is passed through the stdin with an i-time "i" message.

```
proc passData {instr val}
global csoundID
puts $csoundID `` i$instr 0.00 0.0001 $val"
}
```

`csoundID` being the Unix pipe identifier for the previously launched Csound instance, "instr" being the instrument number attached to the controller and "val" being the current value of the slider.

Cecilia also offers a complete text editor to facilitate the elaboration of orchestras and scores. The editor offers services such as syntax and keyword highlighting and direct on-line documentation.

4.2. Cecilia and TclCsound

The new version of Cecilia communicates with the Csound engine via the TclCsound commands only thus eliminating the need for the Unix pipeline. TclCsound's `csChannelIn` Value command eliminates the need for console-driven events to update values.

The improvement of performance from the use of internal piplines is notable but the main benefit is to make Cecilia completely platform independent. This means Cecilia can finally run on the Windows operating systems the same way it runs on UNIX systems. The interface and the synthesis engine run under a single process.

4.3. New Functionalities of Cecilia

It is now possible to make multi-layered arrangements of Cecilia processing modules by simply chaining the output of one module as the input signal to another, effectively paving the way to extremely complex signal processing networks. The interface controls for "stacked" modules are presented in a tabbed window for easy access.

Most Cecilia sound processing modules can now be applied to live input audio streams instead of just sound files.

It is now possible to record, display and edit any gestural controller data entered via MIDI input. The Cecilia grapher window now contains a bezier function editor and a data-reduction method to properly implement a keyframe based automation system. This feature is borrowed from video editing software like Adobe After Effects.

4.4. Cecilia for audio-visual composition

The advent of audiovisual composition as one of the most promising areas of computer assisted creative disciplines is influencing our design decisions. It is now possible to export Cecilia grapher functions as keyframe maps directly to Adobe After Effects, Processing or other image processing software. This feature is expected to be an important advance for the establishment of audio-visual mapping strategies where time varying parameters of the audio stream can be used to control time variant parameters of image processing software.

5. CONCLUSIONS

Csound5, TclCsound and Cecilia are an example of the future development of audio processing software where intuitive and very responsive interfaces are built around powerful, general and highly programmable DSP engines. It is thereby shown that the power of low level signal processing languages is only accessible to non-specialist composers and creators inside responsive interaction contexts, a combination readily provided by TclCsound and Cecilia.

6. REFERENCES

[1] B. Vercoe, *Csound: A Manual of the Audio Processing System*. MIT Media Lab, 1986.

[2] F. R. Moore, *Elements of Computer Music*. Englewood Cliffs, N.J.: Prentice Hall, 1990.

[3] S. T. Pope, "Machine tongues XV: Three packages for software sound synthesis," *Computer Music J.*, vol. 17, no. 2, pp. 23–55, 1993.

[4] J. ffitch, "On the design of Csound 5," in *Proc. 3rd Linux Audio Conf.*, 2005, pp. 37–42.

[5] V. Lazzarini, "Scripting Csound 5," in *Proc. 4th Linux Audio Conf.*, 2005, pp. 50–55.

[6] J. Piché and A. Burton, "Cecilia: A production interface to Csound," *Computer Music J.*, vol. 22, no. 2, pp. 52–55, 1998.