Cmix on Non-UNIX Platforms

Alistair Riddell
Music Department
La Trobe University
amr@farben.latrobe.edu.au

Ross Bencina
Music Department
La Trobe University
rosh@klang.latrobe.edu.au

Abstract

MacCmix and BeCmix are recent ports of Cmix to non-UNIX environments. Both ports make Cmix available to users possibly unfamiliar with the UNIX operating system yet retain much or all of the original Cmix charm. This paper describes some of the interface and implementation details. MacCmix is available from:

1. Overview

Cmix is a toolkit for analysis, synthesis and processing of sound. It has been available for UNIX machines for more than ten years. It has been used, modified and augmented by many computer music practitioners and in the process become one of the "layers" in the computer music world. For further information see Paul Lansky's [1990] overview and history of Cmix.

Two ports of Cmix are described here. The first is to the Power Macintosh environment. It makes Cmix available to Mac users in a way that retains much of the original Cmix environment. The second is to the Be box. This was a more straightforward undertaking because of the Be's POSIX compliance. Documentation for MacCmix can be found at: http://farben.latrobe.edu.au/MacCmix/ with most of the information straight from the original Princeton Cmix distribution.

2. MacCmix

The MacCmix port, undertaken in 1995, necessitated significant alterations to the low level code of Cmix. We begin by outlining the key attributes of Cmix that were interesting challenges during the porting process.

A cross sectional view of Cmix reveals several functional "layers" which give some idea of where the critical porting issues are located. The layers are:

- Compositional Interface
- Utility and Signal processing functions
- Script language dispatch
- Soundfile I/O

The first layer can be independent of basic Cmix operations. This is where the composer decides how to control Cmix functionality. Consequently, this layer was of no immediate concern to us except that the port should in no way hamper the users activities.

The next layer concerns the suite of tools for analysis, mixing and signal processing. The tools are typically coded in C and call the lower level functions. Some of the tools e.g. LPC, have proven difficult to port, primarily because of their dependence on certain low level sound file I/O functions. We are still looking into these.

The final two layers are where the interesting porting issues of Cmix can be found.

The following issues were addressed and for the most part involved changes to the kernel code.

- Convert code to ANSI C
- Convert disk I/O from UNIX to ANSI I/O and EBICS to AIFF format
- Create a shell / window environment
- Applescript replaces Mine

Convert code to ANSI C

Making the Cmix code ANSI standard was important and intended to speed porting to other platforms in the future (there are plans for a version running under win95!). For the non-unix ports it was decided that certain attributes should change to accommodate future extensions. The most important changes deal with the soundfile data structures. A set of arrays, one for each soundfile property, becomes an array of structs, one for each soundfile. Other changes were more cosmetic and often only involved a change in coding style.

ICMC Proceedings 1996 263
A shell/window environment in some form or other, is critical to Cinx. There was no available code that could easily be adapted for standard I/O to a console type window, allowing the application to receive Apple events (a Macintosh IAC protocol employed to execute user functions in MacCinx). A SOUX replacement supporting dynamic Apple event dictionary (AEITE) builds, was implemented. It contains main() and is linked with LibToolShell. The following are some important functions for the window environment:

CMIXInit.c - Contains TSInitTools() which is called by ToolShell at startup time. This calls ug_intro() and profile(). Also contains the AE V ersioning code.
ug_intro.c - contains the ug_intro function, which installs all cmix intrinsics.
ug_intro - Allows information about a function to be stored in the AppleScript data base dictionary.

Disk I/O UNIX to ANSI, EBICS to AIFF

Porting the soundfile I/O was a messy operation since the Cmix I/O system depends on the "booley file" property of BSD file systems. All of the soundfile I/O has been redesigned using standard ANSI functions. You can't have files with holes in them on the Mac, so the new I/O routines actually write silence into the file.

To compound problems, Cmix uses the EBICS soundfile format which is not compatible with the Mac. Many programs use the header manipulation macros (st*) to get at sound file header information. This was replaced by a more direct approach which maintains a structure with all the interesting header data and two functions one two update this struct from disk and one to write it out to disk. Pointers to these functions are stored independently for each soundfile, so theoretically it is possible to support a number of soundfile formats although only AIFF and raw formats are implemented (RAW is useful for dealing with Sound Designer II files).

Changing from EBICSF to generic header manipulation independent (the soundfile type was a good idea, after all most processes don't care about the soundfile format (even if they care about the sample size)). However the present implementation leaves a lot to be desired, and is currently under review for the win95 port.

AppleScript replaces Minc

AppleScript seemed like a really good idea at the time, so a method was devised for calling Cmix user functions from apple script. This worked, except it introduced a number of other problems including:

1. Creating a version that would run on 68k and power macs due to AppleScript floating point representations.
2. Each executable is required to have a unique signature (which is normal on the Mac but not usually as necessary.)
3. And now for the big one:

AppleScript binds message name to message signatures (the underlying representation) using a resource in the target application called the AEITE resource. Is order for the standard, off the street MacCinx user to not have to worry about this when writing their killer Cmix instrument. A system by which this AEITE resource is generated when the program is launched was devised (a modification of the familiar UG_INTRO macro did the trick). Now all is well and good if the AEITE resource exists when you start trying to talk to your app with apple script. But if not, AppleScript gets really confused. There is a way around this (tell AppleScript to explicitly ask the app for its bindings rather than poke around for them) but thus far we have been unsuccessful in getting this to behave correctly.

Fundamental to the problem of using AppleScript is it's means of providing automation for clearly defined functionality. By contrast, one of the conceptual highlights on Cmix is that all you have to do is write a function in C:
int myCmixFun(char**p, int nargs)
{
    /* hi there! */
}

and tell the kernel it exists in your profile:

void profile(void)
{
    UG_INTR("doMyFun","myCmixFun");
}

It is our present belief that AppleScript (or Apple's open scripting architecture) is not well suited to Cmix's late bound dispatch system. We already have a new version on the NeXT using Tcl and it is intended that in future the script interface for portableCmix will be cleaned up so that any scripting system can easily be attached.

Aims for the future:
- make the system more portable
- interface to arbitrary scripting systems
- variety of sound file formats
- more documentation
- lots more comments in the code
- more examples
- more unified suite of tools

portableCmix is very much in its early stages, as with other incarnations of Cmix it will evolve over time and become something different on every user's machine. It is our aim however, to eventually create something which is as much a source of reference to new users as it is a valuable tool to old hands.

3. BeCmix

At the time of writing this article the Be Box was under 1.1.56 version of the OS. Version 1.1.07 (DR7) has substantially more POSIX compliance and therefore Cmix compiles with less alterations. The Be Box has a BASH shell environment and its general interface, for all intents and purposes, feels like a hybrid Mac/Win95/NeXT.

Minc, the original Cmix script language, runs on the Be. The Minc code from the UNIX source compiled without recourse to Bison. Cmix doesn't take advantage of any of the Be's operating system advances like it's fully thread kernel or specific sound driver calls. The Be, like most contemporary computers, has been designed with a view towards real-time processing. While certain Cmix functions can run in real-time, its architecture and philosophy are towards non-real-time sound processing and creation of sound files. Cmix's future probably lies somewhere in between the real-time and non-real-time worlds. But as is well known, creative demands can probably always outstrip processing speeds.

References