On The Design of Csound5

John ffitch
Department of Computer Science
University of Bath
Bath BA2 7AY,
UK,
jpff@cs.bath.ac.uk

Abstract

Csound has been in existence for many years, and is a direct descendant of the MusicV family. For a decade development of the system has continued, via some language changes, new operations and the necessary bug fixes. Two years ago a small group of us decided that rather than continue the incremental process, a code freeze and rethink was needed. In this paper we consider the design and aims for what has been called Csound5, and describe the processes and achievements of the implementation.

Keywords

Synthesis language, Csound.

1 Introduction and Background

The music synthesis language Csound (Boulanger, 2000) was produced by Barry Vercoe(Vercoe, 1993) and was available under the MIT Licence on a small number of platforms. The current author ported the code to the Windows environment in the early 1990s, whereupon a self-defining team of programmers, DSP experts and musicians emerged who have continued to maintain and extend the software package ever since. The original synthesis engine has remained largely unchanged, while a significant number of new operations (opcodes) and table creation routines have been added. Despite various suggestions over the years, the two languages — the score language and the orchestra language — have remained unaltered until very recently, when user-defined opcodes, `if..else` and score looping constructs were introduced.

The user base of Csound is large, and as we have maintained a free download policy we do not know how many copies there are in existence or how many are being used. What is clear from the Csound mailing lists is that the community is very varied, and while some of us think of ourselves as classical "art" composers, there are also live performers, techno and ambient composers, and many other classifications.

The subject of this paper is Csound5, and in particular how its design has evolved from the current Csound. But there are two particular phenomena that have had a direct influence on the need for the re-think.

The first was legal; Csound had been distributed under the MIT licence since 1986, which stipulates some freedoms and some restrictions. The freedoms are expressed as *Permission to use, copy, or modify these programs and their documentation for educational and research purposes only and without fee is hereby granted, provided that this copyright and permission notice appear on all copies and supporting documentation.* There was clarification that this should be taken to allow composers to use it without imposing any restriction on the resulting music. However the licence continues *For any other uses of this software, in original or modified form, including but not limited to distribution in whole or in part, specific prior permission from M.I.T. must be obtained.* When Csound was first made available this was considered a free licence, but with the growth of the Free Software movement, and much wider availability of computers, the restriction stopped developers making use of Csound in larger software systems if they were intending to distribute the resulting system. It also acted to prevent some kinds of publicity, as might be engendered by inclusion in books and magazines. Early attempts to resolve these problems failed, mainly though incomprehension. The publication of Phillips' book(Phillips, 2000) was a further call to address the problem. The change which influenced the whole approach to the development of Csound was the adoption by MIT of the Lesser GNU Public Licence. The *de facto* monopoly allowing distribution was gone.

The second phenomenon was the apparently remorseless improvements in technology. Csound was conceived as an off-line program, rendering a sound description over however long

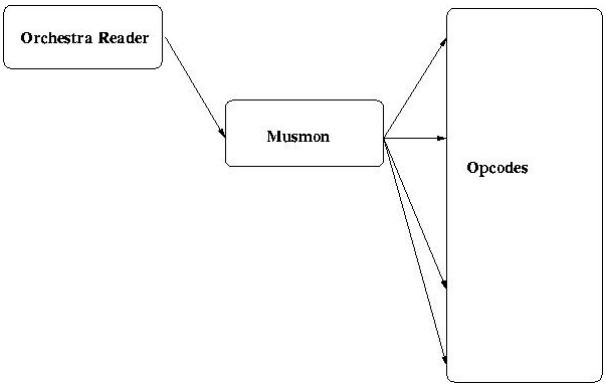

Figure 1: Architecture of original Csound

it took. In the mid 1990s there was a project to recreate Csound for an embedded DSP processor(Vercoe, 1996) as a means of making a real-time synthesis system. This has been overtaken by the increase in machine speeds, and this speed has resulted in the Csound community calling for real-time performance, performer interfaces and MIDI controls. While some users had been wanting this for years, the availability of processors that were nearly capable of real-time rendering made all too clear the shortcomings of the 15- year-old design.

At the end of 2002 we imposed a code freeze to allow the developer community to catch up with their modifications, and in particular to allow larger scale changes to be made on a fixed target. The previous version was still subjected to bug fixes but mainstream development ceased as we moved to Sourceforge and opened up the system even further.

This paper gives one person's view of the system we are building, usually called Csound5, as we froze at version 4.23. As the system is now running largely satisfactorily it is a good time to reflect on the aims of this major reconstruction, and to what extent our aspirations have been matched by our achievements.

2 Requirements

The developers had a number of (distributed) discussions of what was needed in any revision. The strongest requirement was the ability to embed Csound within other systems, be they performance system or experimental research testbeds(ffitch and Padget, 2002). This has a number of software implications. The most significant one is perhaps the need for an agreed application process interface (API) which would allow the controlling program access to some of the internal operations of Csound, and also separate the compilation processes from the execution. Also in the scope of the API is the possibility of adding new opcodes and generators which have access to the opcode mechanisms, memory allocation, navigation of internal structures and so on.

Related to the requirement for a documented software interface is a call to make Csound reentrant. This would allow multiple uses both serially and in parallel. The original code was written with no thought for such niceties, and there is a plethora of static variables throughout the system. Removing these would be a major step towards re-entrance, and encapsulating the state within a single structure was the proposed solution, a structure that could also carry parts of the API.

A possible lesser goal was to improve the internal reporting of errors. The original system set a global variable to indicate an initialisation error or a performance error, and this is checked at the top event loop. A simpler and more respectable process is for each initialiser and operator to return an error code; such a system can be extended to make use of the error codes.

Csound originally generated IRCAM format sound files, and AIFF. Later WAV was added and some variants of AIFC. The code was all *ad hoc* and as audio formats are continually being developed, it seemed an ideal opportunity to capitalise on the work of others, and to use an external library to provide audio filing.

In a similar way the real-time audio output is specially written for each platform, and maintaining this reduces the time available for development and enhancement. Since Csound was written, cross-platform libraries to encapsulate real-time audio have been developed, and while using an external library for files it seemed natural to investigate the same for sound.

Another aspect where there was platform-dependent code is in graphics. Csound has been able to display waveforms and spectral frames from the beginning, but there are a large number of optional files for DOS, Windows, Macintosh, SGI, SUN, X, and so forth. Using a general graphical system would move this complication into someone else's hands. It would also be useful if the graphical activity were made external, using the API, so a variety of graphical packages could be used in a fashion like embedding. This leads to the idea of providing a visible software bus to communicate between the Csound engine and the wider environment.

The last component where an external library could assist is in MIDI. There have been complaints about the support for MIDI for a long time, and so in any reconstruction it was clearly something that should be addressed.

The last major component that is in need of reconstruction is the orchestra parser. The original parser is an *ad hoc* parser very reminiscent of the late 1970s. It is hard to modify and there are bugs lurking there that have evaded all attempts to fix. If a new parser were to be written it could sidestep these problems and also allow things like two-argument functions, which have been requested in the past. Another possible outcome from a new parser might be the ability to experiment with alternative languages which maintain the underlying semantics. That might also incorporate the identification of a parser API.

In all this design we were mindful that Csound was and must remain a cross-platform synthesis system, and should behave the same on all implementations. It would also be convenient if the building system were the same or similar on all platforms, and installation should be simple — accessible to users at any computer-literate level.

The other overriding requirement is that the system must not change externally, in the sense that all old music pieces must still render to the same audio. We can add new functionality, but visible things must not be removed.

3 Implementation

The previous section described the desired features of the new Csound. But they are wishes. In this section we consider the translations of these aspirations to actual code.

The API is largely the work of Gogins, but there is a number of basic concepts in the solution. The implementation is by a global structure that is passed as an argument to most functions. Within the structure there are at least three groups of slots. The first group incorporates the main API functions; functions to control Csound, such as Perform, Compile, PerformKsmps, Cleanup and Reset. There are also functions in this structure to allow the controlling program to interrogate Csound, to determine the sampling rate, the current time position and so forth. These functions are also used by user-defined opcode libraries to link to the main engine. The last group are the state variables for the instantiation of Csound.

The transition to allowing a re-entrant system is largely one of moving static variables into the system-wide structure. Code simplicity is maintained by creating C macros so access can be via the same text as previously. By adding an additional argument to every opcode of this environment structure a great simplification of much of the code is achieved, especially for user-defined opcodes, as described in more detail below (section 4).

Every opcode now returns an error code, or zero if it succeeded. This is a facility that has not been exploited yet, but it should be possible to move more of the error messages from the main engine, and incidentally to facilitate internationalisation.

The decision to use an external library for reading and writing sound files was an easy one; what was less easy was deciding which one to use. A number were considered, both the small and simple, and the all-embracing. The one we chose was Libsndfile (de Castro Lopo, 2005). The library is subject to LGPL, but the deciding factor was the helpful attitude of the author. We have not regretted this decision, and it was moderately easy to replace the complex accumulation of AIFF, AIFC and WAV with the cleaner abstraction. The hardest part was that Libsndfile works in frames and Csound has been written in samples or sometimes bytes. Of particular note was the removal of many lines of code that dealt with multiple formats (alaw, μlaw, signed and unsigned...).

There seemed less choice with the real-time audio library; PortAudio (Bencina and Burk, 2005; Bencina and Burk, 2001) seemed obvious. As the library was in transition from version 18 to 19 we decided to look ahead and use v19. This has been a more problematic decision. For example Csound is written with a blocking I/O model for audio, but to date of writing this is not implemented on all platforms, and we are using a hybrid, implementing blocking via callbacks and threads on some systems, and simple blocking I/O on others. There have even been suggestions that we abandon this library as it has not (yet) delivered the simplicity we seek. I think this can be overcome, and the decision was correct, but there are clearly problems remaining in this area.

The companion to PortAudio in the PortMusic project(Por, 2005) for MIDI is PortMIDI(Dannenberg, 2005). This was the obvious choice to support MIDI. The software mod-

els are fairly far apart but it has been incorporated. What we do not seem to be able to find is a library for file-based MIDI. At present we are continuing to use the original Vercoe code, with what looks like duplication in places. This needs to be investigated further.

There is a surfeit of graphical toolkits, at many levels of complexity. Based on previous experience, both outside Csound and inside with CsoundAV(Maldonado, 2005), FLTK was chosen. This is simple and light-weight. There are undoubtedly faster libraries, but graphics performance is not of the essence and the simplicity is worth the loss. A drawback is that this is a C++ library, whereas Csound is still at heart a C program. However in the medium term I still intend that graphics should be moved out of the core Csound and we should use the API and software bus.

A contentious issue (at least within our developer community) has been a framework for common building. For most of the life of Csound there have been three major builds, for Linux, Windows and Macintosh. The Linux and Unix system use a hand crafted makefile; on Windows a Microsoft Visual C++ IDE was used and on Macintosh the Codewarrior IDE. The redesign of Csound coincided with the acceptance of OSX on the Macintosh, and the availability of the MinGW package for Windows. This suggests that it should be possible to have a common tool-chain. Initial experience with the GNU tools (automake, autoconf etc) was highly negative, with incompatibilities between platforms, and between different releases of Linux. We are now using SCons(SCo, 2005) which is a Python-based building system which we have found to work cleanly on our three major platforms, and to have sufficient flexibility.

The first implementation of a software bus has been made, by offering an arbitrary number of uni-directional audio and control buses. This facility remains to be exploited.

The most problematic area of the implementation is the parser. A Flex-based lexer and a Bison parser have been developed[1] and these implement most of the current Csound language. The problem of joining this front-end into the internal structures remains as a major task that has not yet been attempted. The design of the parser will allow user-defined opcodes as is essential, as well as functions of one or more arguments. The main incompatibilities are in the enforcement of functions as functions, which is not in the original system. It does however mend known bugs in the lexing phase, and also makes better use of temporary variables.

[1]The parse is not based on the earlier Bernardini parser, but created with the support of Epigon Audiocare Pvt Ltd

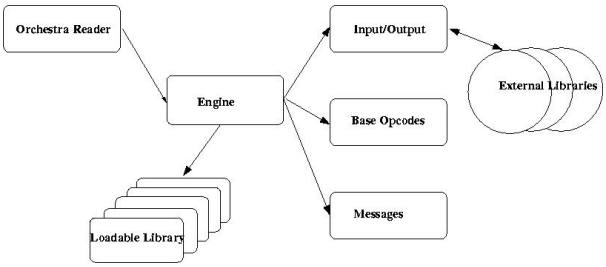

Figure 2: Architecture of Csound5

4 User Defined Libraries

One reason for the redesign was to allow third parties to provide new opodes, either as open source or as compiled libraries that can be loaded into Csound. The user opcodes are compiled into .DLL or shared libraries, and the initialisation of Csound loads libraries as required.

User libraries were introduced in Csound4, but in Csound5 they have been extensively developed. We provide C macros to allow the library to be written in much the same way as base opcodes, and proforma structures to link the opcodes into the system. We have also recently made it possible to have library-defined table generators as well. The macros wrap the usual internal Csound functions as called via the global environment structure.

To prove that the mechanism works, many of the opcodes were remade as loadable code. The final decision as to which opcodes will be in the base and which loadable is not settled, but the overall structure of Csound is now changed from the architecture of figure 1 to that of figure 2. With this architecture we hope that clearer separation will make maintenance simpler.

5 Experience

In many ways it is too early to make informed judgements on the new Csound. On the other hand the system has been in a running state for many months, and on at least the Linux platform it is usable. Despite some rough edges it renders to both file and audio, and there are no appreciable performance issues.

The use of Libsndfile has been a very positive experience on all platforms. PortAudio has

caused some problems; with ALSA on Linux it is acceptable, but there have been latency problems on Windows and a number of ongoing problems on OSX, with lack of blocking I/O and an apparent need for multiple copying of audio data. There are enough indications from the PortAudio development community to say that this will work to our advantage eventually. It is still too soon to comment on the MIDI components.

There are still questions that arise from graphics and in particular the control of multiple threads. I believe that the solution is to use the software bus and outsource graphical activity to a controlling program. The graphics does work as well as it did on Csound4, but problems arise with the internal generation of GUIs for performance systems.

The code freeze has had a number of minor positive effects; the code has been subjected to coherent scrutiny without pressures for releases. Multiple identical functions have been combined, and many small coding improvements have been made, for both stylistic and accuracy reasons.

The current state is close to release. It might be possible to release before the incorporation of the parser, but this would be a disappointment to me. The other aspect that may delay release is documentation. The manual still needs updating. Basic information on the system already exists.

The decision to use SCons has proved excellent. It is working on Windows and OSX as well as the usual development Linux platforms.

6 Conclusions

In this paper I have described the thoughts behind the creation of the next incarnation of Csound. Evolution rather than revolution has been the key, but we are creating an embeddable system, a system more extensible than previously, and with clear component divisions, while preserving the operations and functionality that our users have learnt to expect. By concentrating on an embeddable core I hope that the tendency to create variants will be discouraged, and from my point of view I will not have to worry about graphics, which interests me not at all!

While the system has remained a cross-platform one, development has been mainly on Linux, and we have seen great benefits from all the tools there. When Csound5 reaches its distribution time soon, the musical community will also see these benefits.

7 Acknowledgements

My thanks go to everyone on the Csound Development mailing list for all their input, comments and reports, and also to all the Csound users who made it so clear what they wanted. Particular thanks are due to Michael Gogins for his insistence on a sane API, and to Richard Boulanger who has been a driving force behind me in the development and maintenance of Csound.

References

Ross Bencina and Phil Burk. 2001. PortAudio – an Open Source Cross Platform Audio API. In *ICMC2001*. ICMA, September.

Ross Bencina and Phil Burk. 2005. PortAudio. `http://www.portaudio.com/`.

Richard Boulanger, editor. 2000. *The Csound Book: Tutorials in Software Synthesis and Sound Design*. MIT Press, February.

Roger B. Dannenberg. 2005. PortMIDI. `http://www-2.cs.cmu.edu/~music/portmusic/portmidi`.

Erik de Castro Lopo. 2005. Libsndfile. `http://www.mega-nerd.com/libsndfile/`.

John ffitch and Julian Padget. 2002. Learning to play and perform on synthetic instruments. In Mats Nordahl, editor, *Voices of Nature: Proceedings of ICMC 2002*, pages 432–435, School of Music and Music Education, Göteborg University, September. ICMC2002, ICMC.

Gabriel Maldonado. 2005. Csoundav. `http://www.csounds.com/maldonado/download.htm`.

Dave Phillips. 2000. *The Book of Linux Music and Sound*. No Starch Press. ISBN: 1886411344.

2005. PortMusic. `http://www-2.cs.cmu.edu/~music/portmusic/`.

2005. SCons. `http://www.scons.org/`.

Barry Vercoe, 1993. *Csound — A Manual for the Audio Processing System and Supporting Programs with Tutorials*. Media Lab, M.I.T.

Barry Vercoe. 1996. Extended Csound. In *On the Edge*, pages 141–142. ICMA, ICMA and HKUST. ISBN 0-9667917-4-2.