# OpenMusic on Linux

Anders Vinjar, Jean Bresson

# OpenMusic on Linux

**Anders VINJAR***
Composer
Norway
anders.vinjar@bek.no

**Jean BRESSON**
STMS lab
IRCAM-CNRS-UPMC
1, place Igor Stravinsky
75004 Paris, France
jean.bresson@ircam.fr

## Abstract

We present a recent port of the OpenMusic computer-aided composition environment to Linux. The text gives a brief presentation of OpenMusic and typical use-cases of the environment. We also present a short history of its development, and mention previous attempts at porting it to Linux. The main technical challenges involved with developing the current Linux port are discussed, as well as solutions to these. We end the paper by outlining some possible areas for future work.

## Keywords

OpenMusic, Computer-Aided Composition (CAC), Linux, Common Lisp, Visual Programming, JACK.

## 1 Introduction

OpenMusic (OM) is an environment for computer-aided music composition designed as a domain-specific visual programming language.[1] It allows composers to write and run programs to transform or generate data, with interactive access to the input or output musical structures.

Before 2013 OM was developed and distributed on OS X and Windows platforms only, despite various attempts at porting the environment to Linux. In this paper we present a new, fully functional port of OM to Linux.

In Section 2 we will introduce the OpenMusic environment, first form a general point of view, and then discuss some particular aspects such as the user interface and the external dependencies of the environment. We also give a quick history of the development of OM, and previous attempts at porting the environment to Linux.

Section 3 describes our current implementation choices and the state of the Linux port. We conclude with a number of perspectives and areas for future work.

* This work is supported by BEK - Bergen Center for Electronic Arts.

[1] http://repmus.ircam.fr/openmusic/

## 2 OpenMusic

### 2.1 A visual programming environment for computer-aided composition

OM is a visual programming environment dedicated to music processing and composition [Assayag et al., 1999]. It implements the main features of the Common Lisp language (abstraction, higher-order functions, recursion, iterations etc., see [Bresson et al., 2009]),[2] as well as object-oriented programming [Agon and Assayag, 2003] and constraints programming [Rueda et al., 1998].

OM is primarily an environment for work in computer-aided composition (CAC). It is also used for musicological tasks like analysis, modeling or statistics, as well as pedagogical work in composition studies or music theory [Bresson et al., 2011]. The environment comes with a rich set of tools and libraries aimed at composition, analysis, DSP and other musical/extra-musical domains.

The aim of a CAC application is to aid the user in typical composition tasks like generating, representing and manipulating musical material in adequate ways, handling musical form as material develops towards a finished piece of music. Contemporary composition is a rather ill-defined activity,[3] and a good CAC tool is one with abilities to adapt well to human artistic processes, whatever those may be. Indeed, composers seldom follow strict plans for too long. More common is perhaps making up a class of material, generate versions on that, develop this again further towards some more complex structures, go back to change some-

[2] Functional programming and Lisp in particular has a strong background and tradition in the computer music history. It has proven to be relatively easy to learn, understand and use by composers.

[3] *"CAC is in effect making the computer carry out thought processes previously carried out in human brains"* (Miller Puckette, preface to *The OM Composer's Book vol. 1* [Puckette, 2006]).

thing in an earlier stage, to arrive at yet another set of variants, and so on.

Interactive (visual) programming languages are well suited environments for composers to work efficiently and creatively with computers. The graphical environment in OM provides an interactive patch-based, data-flow approach, making it easy and intuitive to get going. Figure 1 shows a basic patch window generating a series of chords.

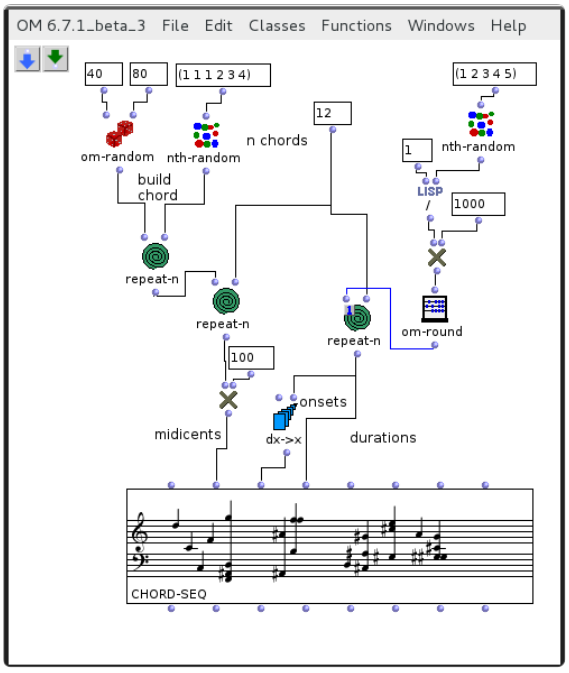


Figure 1: Graphical programming with OM.

OMs dynamic environment supports the kinds of interactive work-flows often preferred by composers, by allowing the user to evaluate and store output at arbitrary stages in the program-flow, and easy stash away variants for any kind of musical material or data.

OM is also a full-blown environment for Lisp-programming, with access to the expected REPL[4], special editors for Lisp code, cross-referencing and look up in source-code, and interactive help-system. These tools provide an environment for composers and programmers to develop new ideas and specialized creative work-flows, using graphical programming, textual programming, or any combination of these. In OM there is no particular separation between a function defined as an abstraction in a patch box or one defined by evaluating Lisp code: both are accessible as graphical objects in the user-interface.

[4]REPL = Read-Eval-Print loop.

Reports of some composition tasks and approaches carried out with OpenMusic are available in [Agon et al., 2006 2008].

## 2.2 User Interface

As a visual programming environment, OM is highly concerned with graphical user interfaces (GUI). To a large extent, this aspect determines the choices of platforms and frameworks that may be used for development.

While programming in OM, the user inserts and manipulates graphical objects in patch windows, dragging connection-lines between inlets and outlets of boxes, and thus establishing the data-flow of a musical program. These graphical objects may represent simple operators, abstractions or sub-patches, or more complex data like lists, arrays, break-point functions etc.

As part of the graphical programming environment, OM provides advanced editors for visualization and manipulation of data such as musical scores, break-point functions, audio waveforms and some other data types. Figure 2 shows the *sound*-editor, and Figure 3 shows editors for various other kinds of musical data.

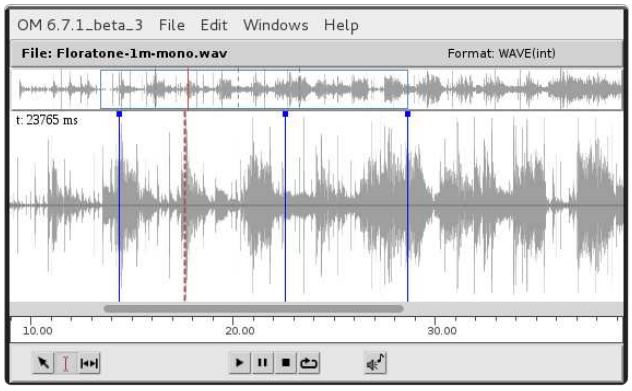


Figure 2: Editor for the OM `sound` object.

OM also has a programmable graphical timeline editor termed the "Maquette", where everything else which lives in OM – functions, musical data, connections – can be placed and manipulated, either manually or as a result of evaluating some code.

## 2.3 Development history – Previous ports and platforms

OpenMusic is a descendant of the Patchwork environment [Laurson and Duthen, 1989], one of the pioneering visual programming systems dedicated to computer-aided composition.

The first release (C. Agon and G. Assayag, IRCAM) was built in 1998 on Mac OS using

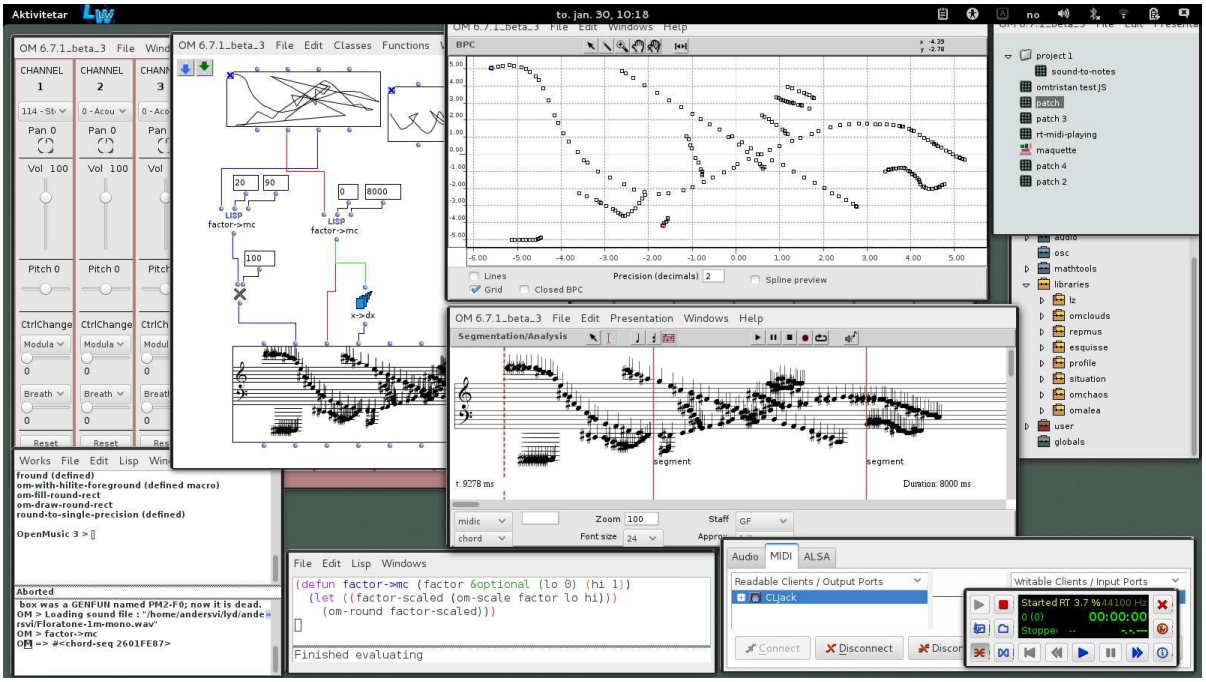

Figure 3: OM environment: musical data and editors.

Digitools' Macintosh Common Lisp (MCL). The graphical programming system was designed as a full meta-programming framework, implementing functional programming concepts and interactions, on a strong base of object-oriented programming and CLOS (Common Lisp Object System [Gabriel et al., 1991]). OM also introduced a number of new concepts concerning for instance handling and manipulation of objects, or unified representation of time structures.

In 2003, OM 4 was ported to OS X, and a first Linux port [Sarria and Diago, 2003] was carried out at IRCAM in the framework of the AGNULA European project [Déchelle and Tisserand, 2003], using CMUCL[5] and Gtk+[6].

Released in 2005, OM 5 [Bresson et al., 2005] was a multi-platform version of OM developed on Mac (using MCL) and on Windows (using Allegro CL[7]). The OM 5 sources were clearly divided into a platform-independent kernel built on top of an abstract graphical/system-dependent API inspired by the MCL toolkit (MCL and ACL versions of this API were then implemented and loaded depending on the targeted platform).

A second port of OM on Linux was initiated in 2006 based on the OM 5 sources, using SBCL[8] and a new implementation of the OM API for Gtk+ and CLG[9] (Common Lisp GTK bindings). Unfortunately this project was never carried to its end.

In 2006, Digitools announced discontinuation of MCL development and support on Mac, due to the switch to Intel processors. The LispWorks environment was chosen as a replacement, providing a reliable IDE with a common cross-platform API (CAPI) compatible with the main graphic toolkits for Mac, Windows and Linux, as well as some other OS-es.

In 2008 OM 6 based on LispWorks was released for OS X and Windows.

### 2.4 External dependencies

OM is dependent on audio I/O systems due to its musical orientation. It is important for composers to be able to load audio or MIDI files, and convert/process the contained data in the visual programming framework. Playback and rendering of generated musical material (score/MIDI or audio) is an essential feature of the environment, and complex scheduling issues can arise when dealing with multiple simultaneous sources and audio/MIDI material interactively. Still, OM is not by nature a real-time environment and the audio performance requirements remain relatively moderate

[5] http://www.cons.org/cmucl/
[6] http://www.gtk.org/
[7] http://www.franz.com/products/allegrocl/
[8] http://www.sbcl.org/
[9] http://sourceforge.net/projects/clg/

as compared to real-time scheduling or audio processing systems.

Low-level MIDI formatting and scheduling was traditionally supported in OM by the MidiShare[10] system [Orlarey and Lequay, 1989]. The audio support, initially limited to a few functions interfacing with the Apple QuickTime library, was replaced in OM 5 by a more advanced and multi-platform audio support developed on top of the LibAudioStream[11] library.

Besides MIDI for file I/O, support for exporting other score-type data is provided either by using built-in code or by using external libraries. OM can export to some much used file formats for sheet-music, like LilyPond[12] and MusicXML[13]. OM also features support for connections to technologies like SDIF [Schwartz and Wright, 2000] (standard format for interchange of sound description data), OpenGL (display of 3D objects in OM editors) and OSC [Wright, 2005] for inter-application communication, although none of these are strictly required to get the visual programming environment running.

OM communicates with external libraries via Common Lisp foreign function interfaces (FFI). OM either uses the FFI provided by the Lisp implementation, or the CFFI system [Bielman and Oliveira, 2013], a common FFI wrapper compatible with several Lisp implementations.

### 2.5 Distribution and licensing

While the first OM releases were distributed commercially along with other IRCAM software, since OM 6.4 (2011), the compiled OM application has been available free of charge for all platforms.

The source-code has always been freely available under the GNU Public License.

All source-code and external libraries required for building OM are open-source. However, to build and save the actual distributed image a LispWorks Professional or Enterprise license is necessary at this moment.

## 3 Towards OM on Linux

Several previous efforts to port OM to Linux over the years suggest a real interest, and OMs main technological dependency (a professional ANSI Common Lisp implementation with interfaces to graphical toolkits, MIDI and audio libraries) has been available to Linux developers for long. Hence, it is legitimate to ask why these previous ports did not succeed too well?

First, the Musical Representations team developing OM at IRCAM use Mac OSX as main development platform. As a consequence, porting OM to the attempted environments (CMUCL or SBCL w. Gtk) means rewriting all the graphical dependencies using foreign toolkits or alternative APIs.

Audio and MIDI support has also been an issue in previous Linux ports. Although MidiShare worked fine with earlier Linux-versions,[14] it is no longer maintained and kept compatible with newer releases of the Linux kernel. Moreover, OM MIDI dependencies were since the early releases packed together and integrated with the kernel code, making it difficult to replace.

As OM has evolved, work towards a more modular structure has been an explicit aim, at least since 2005 [Bresson et al., 2005]. Recent developments have gradually made the code more durable and resistant towards changes in compiler-implementations. This tendency both motivated and greatly helped the work with the current port, making it possible to develop alternative solutions for e.g. MIDI I/O, separated from those for the kernel, graphics or audio I/O.

### 3.1 OM 6.8 on Linux: Current State

When starting this project, a separate Linux development-branch of the source-tree for OM 6.7 was set up, making it possible to get up to speed on Linux without halting further development on the main-branch. At a certain stage the separate Linux-branch was re-integrated with the main source-code. Further development of the application, and delivery of images, is now based on the same source-tree for all three supported platforms – Linux, OS X, Windows – with only a few specializations, mainly in the graphics-code, to account for differences across platforms.

The present Linux development is based on OM 6.8 and LispWorks 6.1. The choice of LispWorks (a commercial Lisp compiler and IDE) for porting OM to Linux is entirely pragmatic: as mentioned, LispWorks provides a common API across graphical toolkits (Gtk+, Motif, Cocoa, Windows), and since this library is

[10] http://midishare.sourceforge.net/
[11] http://libaudiostream.sourceforge.net/
[12] http://www.lilypond.org/
[13] http://www.musicxml.com/

[14] MidiShare was also used in e.g. Common Music [Taube, 1991].

already used by the OM developers, only moderate adaptions to the existing OM API are required in order to get it working with Linux.

As suggested, this port of OM to Linux can be seen as the most recent in a series of steps towards making OM more modular. Solutions which work across platforms are presumably also less vulnerable to changes in compilers or toolkits in use. While developing Linux-compatible substitutes for previous code, general and platform-independent solutions were looked for. In particular, most external dependencies have been made optional, so that OM can run without some specific features, if dependencies are not found, not loaded or not available for a specific environment or platform.

### 3.2 Audio and MIDI I/O: JACK

To substitute the low-level I/O systems for MIDI and audio, some alternative approaches were programmed and tested. The OM "player" system was rewritten in OM 6.7 as a modular API, making it easier to substitute or switch the default playback-engines with alternative audio or MIDI players.

MIDI messages and Standard MIDI Files can now be parsed and formatted using the Common Lisp MIDI library[15], and E. de Castro Lopo's `libsndfile` is used to access audio files on disk. The default playback-engines used in the Linux version of OM depend on `libjack`. Scheduling and real-time I/O of audio and MIDI between OM and hardware-ports (or between OM and other applications) have been programmed as a CL-based JACK[16] client.

Other test-case playback-engines were also developed, for instance with SuperCollider (Audio-MIDI I/O, scheduling) using OSC communication, with a CL-based FluidSynth[17] server (MIDI) controlled through Lisp-code, and with external MPlayer-processes[18] (audio) controlled from a sub-shell. These clients work fairly well, and could be used as examples for users or developers wanting to plug in other playback-engines or I/O systems. An alternative audio player has been developed for instance using `sox`[19] as part of the *OM-Sox* library.[20]

[15] http://www.doc.gold.ac.uk/isms/lisp/midi/
[16] http://jackaudio.org/
[17] http://www.fluidsynth.org/
[18] http://www.mplayerhq.hu/
[19] Sound eXchange – sox.sourceforge.net/
[20] M. Schumacher, *OM-Sox*: http://sourceforge.net/projects/omsox/

### 3.3 Libraries

OM comes with a number of specialized tools and libraries aimed at composition, analysis, DSP and other musical tasks. Some of these are "official" libraries: either distributed and maintained as part of the main OM package, or used to integrate OM with external tools such as Csound or some of the DSP engines available from IRCAM (SuperVP, PM2, Spat, Diphone, Chant).[21] A rich set of 3rd-party libraries, maintained by individual composers or developers, are also available for download,[22] together with simple how-tos for adding external libraries (see Figure 4).

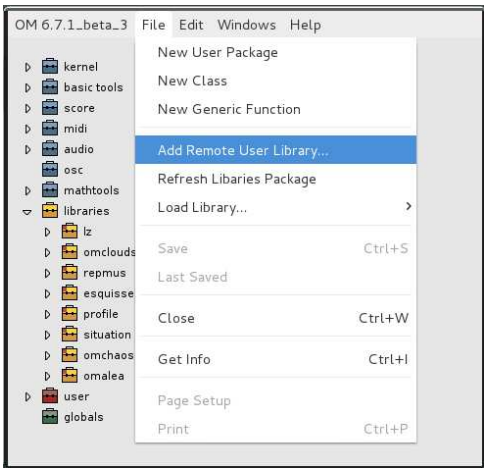


Figure 4: OM: Packages library.

Libraries are written as standard Lisp code, so they generally work well with the Linux-port. However, those depending on platform-specific external tools (e.g. OM-Spat) are not very useful at the moment.

### 3.4 Source code and packaging

The OM source code is distributed as part of the application.[23] The environment features run-time introspection and provides an easily accessible cross-reference for all OM-specific classes and methods. Lisp code may be edited and evaluated interactively, or loaded from files, and may be used to specialize or modify built-in functionality.

To compile a fresh OM from sources, access to LispWorks is necessary. For this reason OM is always made available as a pre-built image, one for each platform-type. At the time of this

[21] http://forumnet.ircam.fr/product/openmusic-libraries/
[22] http://repmus.ircam.fr/openmusic/libraries
[23] http://repmus.ircam.fr/openmusic/sources

writing, the Linux version is developed and maintained on a system running Fedora 19. Currently, OM is available as a `RPM`-package containing the image and all sources, which will install the binary and all sources in the usual places. Several users have adapted the RPM-packages to `dpkg`-based systems, seemingly without any serious issues, and also shared how-tos and experiences on the OpenMusic forums[24].

Patches are usually distributed as Lisp files. These can be dynamically loaded by the user or be automatically sourced on application start by placing the files in a predefined location.

## 4 Conclusions – Future works

The first beta-release of OM-Linux was made available for download and presented at the IRCAM Forum workshops in November 2013, after having been tested and used by developers and users for some time.[25]

The previous Linux ports missed their potential audience and lacked support and follow up by the Linux developers and composer's community, presumably due to a number of obstacles and difficulties that we have tried to outline in this paper. Our hope is that this project will overcome most of these problems.

A working Linux-version of OM is useful for end-users, and Linux-developers may well find good ways to integrate OM with other applications through e.g. libraries, or to develop OM further. The stabilization of the GUI API may also help to lessen dependencies on LispWorks in the future for all platforms, and make alternative open-source solutions possible. In this effort, starting out from a functional version on Linux may be valuable.

Since the current code uses one common source-tree, the Linux-port is relatively robust and sustainable across changes in compiler-implementations or frameworks for graphics and I/O. It also minimizes the work involved in maintaining compatibility across platforms for the same application.

The features introduced to make the sources compile and run on Linux, as well as the new developed support e.g. for audio and MIDI are of a general kind, and potentially useful across platforms. The CL-based MIDI-library, as well as the JACK-client and callback setup for MIDI and Audio, are now possible alternatives also for OS X and Windows.

Further work will be done in the near future to integrate other CL-based composition and DSP-tools such as Common Music, or Common Lisp Music (CLM[26]) [Schottstaedt, 1994], and to extend the existing set of libraries connecting OM with open-source software like e.g. SuperCollider or LilyPond. While the JACK-client is currently useful, real-time robustness can still be improved, and other audio-libraries or backends may also be supported in the future.

## 5 Acknowledgments

The authors would like to thank Trond Lossius and BEK (Bergen Center for Electronic Arts) for their support on this project. This work is also partly developed in the framework of the French ANR project with reference ANR-13-JS02-0004-01.

## References

C. Agon and G. Assayag. 2003. OM: A Graphical Extension of CLOS using the MOP. In *Proc. of International Lisp Conference*, New York, USA.

C. Agon, G. Assayag, and J. Bresson, editors. 2006–2008. *The OM Composer's Book (2 volumes)*. Delatour/IRCAM.

G. Assayag, C. Rueda, M. Laurson, C. Agon, and O. Delerue. 1999. Computer Assisted Composition at IRCAM: From PatchWork to OpenMusic. *Computer Music Journal*, 23(3).

J. Bielman and L. Oliveira. 2013. CFFI home page. http://common-lisp.net/project/cffi/.

J. Bresson, C. Agon, and G. Assayag. 2005. OpenMusic 5: A Cross-Platform Release of the Computer-Assisted Composition Environment. In *Proc. 10$^{th}$ Brazilian Symposium on Computer Music*, Belo Horizonte, MG, Brasil.

J. Bresson, C. Agon, and G. Assayag. 2009. Visual Lisp/CLOS Programming in OpenMusic. *Higher-Order and Symbolic Computation*, 22(1).

J. Bresson, C. Agon, and G. Assayag. 2011. OpenMusic – Visual Programming Environment for Music Composition, Analysis and Research. In *Proc. ACM MultiMedia*, Scottsdale, AZ, USA.

[24] http://forumnet.ircam.fr/user-groups/

[25] A review of this Linux port has been posted by D. Philips on LWN in November 2013, see https://lwn.net/Articles/574593/.

[26] https://ccrma.stanford.edu/software/clm/

F. Déchelle and P. Tisserand. 2003. The AGNULA project. Free software developments at IRCAM. In *International Workshop on Free Software and Research*, Compiègne, France.

R. P. Gabriel, J. L. White, and D. G. Bobrow. 1991. CLOS: Integration Object-oriented and Functional Programming. *Communications of the ACM*, 34(9).

M. Laurson and J. Duthen. 1989. Patchwork, a Graphic Language in PreForm. In *Proc. International Computer Music Conference*, Ohio State University, USA.

Y. Orlarey and H. Lequay. 1989. MidiShare : a Real Time multi-tasks software module for Midi applications. In *Proceedings of the International Computer Music Conference*, Ohio State University, USA.

M. Puckette. 2006. Preface. In C. Agon, G. Assayag, and J. Bresson, editors, *The OM Composer's Book .1*. Delatour/IRCAM.

C. Rueda, M. Laurson, G. Bloch, and G. Assayag. 1998. Integrating Constraint Programming in Visual Musical Composition Languages. In *Proc. European Conference on Artificial Intelligence*, Brighton, UK.

G. Sarria and J. F. Diago. 2003. OpenMusic for Linux and MacOS X. Technical report, IRCAM.

Bill Schottstaedt. 1994. Machine Tongues XVII: CLM: Music V meets Common Lisp. *Computer Music Journal*, 18(2):30–37.

D. Schwartz and M. Wright. 2000. Extensions and Applications of the SDIF Sound Description Interchange Format. In *Proc. International Computer Music Conference*, Berlin, Germany.

H. Taube. 1991. Common Music: A Music Composition Language in Common Lisp and CLOS. *Computer Music Journal*, 15(2).

M. Wright. 2005. Open Sound Control: An Enabling Technology for Musical Networking. *Organised Sound*, 10(3).