An abstraction's creation arguments may be either numbers
or symbols. Gory details are inside:

dollarsign2 three 4

updated for Pd version 0.34

The Search for Usability and Flexibility in Computer Music Systems

Günter Geiger

1. Introduction

> "He (the user) would like to have a very powerful and flexible language in which he can specify any sequence of sounds. At the same time he would like a very simple language in which much can be said in a few words." – Max Mathews

Computer Music, that is, music created on a computer with the help of mathematical formulas and signal processing algorithms has always been a challenging task, one that traditionally has been difficult to handle both by humans, because of the need of a fundamental understanding of the algorithms, and by machines, because of the demands on computational power when implementing the algorithms.

The first person who took this challenge and came up with a system for computer generated music was Max Mathews, working at the Bell Laboratories in the late 50s. We are about to celebrate the 50th birthday of computer music, and we therefore have to ask ourselves what our computer music systems (CMS) evolved into, and where to go from here.

Nowadays the processing power of everyday computer systems can easily handle most of the problems of computer music: can we say that the human factor in computer music is equally developed? Has the software matured in a way similar to the increase in hardware power? I will attempt to outline the levels of abstraction that we have reached and describe the established metaphors with a sketch of the main properties of computer music systems and a historical overview of the most influential systems and their implementation.

Intuitivity, Usabilility and Efficiency

The goal of software is to facilitate tasks for humans. This means that the achievment of the task is the minimal requirement, the quality of the software is measured by the "efficency" with which a specific task is solved.

During the last 50 years the definition of the task in computer music has broadened considerably, today taking into account almost everything that can be done with computers and music. In the beginning, producing digitally generated sound in realtime was unthinkable, so the systems did not take interactivity and other realtime aspects into account.

The broadness of the field makes it difficult to come up with one general model for computer music, and despite the evolution of computer music during the last 45 years, computer musicians today still fall back into using low level languages for expressing their algorithms.

A useful computer music system should therefore offer a reasonable level of abstraction, but it should still offer ways for expressing new concepts. Beside the abstraction of well know algorithms, computer music languages are also concerned with the performance of these algorithms.

2. From Music I to GROOVE

The era of computer music started in 1957 when Max Mathews wrote the first software synthesis program called "Music I". It ran on an IBM 704 Computer at Bell Laboratories were Mathews was working.

At that time programs where mainly written in assembler for performance reasons and because of the lack of real high level languages, so Music I, the first incarnation in a series of programs known today as Music-N, was also written in assembler. Mathews was going through iterations of updates on the Music I program, which eventually led to a version that was called "Music V", written in the high level language Fortran this time.

Music V can be considered as the breakthrough of the line of programs referenced as Music-N. Mathews' book about Music V[15] is still one of the main reference works about the structure of computer music systems, and the Music V system produced and still produces systems that follow its basic principles. One could say that Music V defined the structure of computer music programs and influenced and formed the way we think about computer music, especially if we talk about software synthesis practice.

At the time when Music V was written, computers were not fast enough to calculate audio signals in real time, so Music V was a program that was used to calculate sound files in non-real-time. In the early 70s, Max Mathews started another project called the "GROOVE" system[27], the first system that was designed for realtime control of synthesizers.

Other important works from the early era include the "MUSICOMP"[21] system by Lejaren Hiller, author of the first published computer music work in 1958, the "ILLIAC suite". This system was a composition system, and not a sound synthesis package.

The "MUSIGOL" system was a sound synthesis package based on the ALGOL language, one of the first high level languages that appeared during that time.[25]

3. Music-N Descendants

The Music series of programs was a success, and the fact that Mathews shared the code with other institutions, along with the need to port the code to different computer architectures, led to several extended versions of Music-N.

One line that came out of this process was the "Music4B" and "Music4BF" from Godfrey Windham and Hubert Howe. The F in Music 4BF Stands for Fortran, which means that parts of that system were implemented in Fortran too.

Building on Music 4B Berry Vercoe wrote a CMS for the IBM System/360 which he called "Music 360" in 1968, later on in 1974 a program for the "PDP-11: Music 11". A direct follow up of Music 11 was "CSound", coming in 1985, today maybe the best known and most used incarnation of a MUSIC-N system.

At Stanford, yet another advanced incarnation of Music V was developed, called "MUS10", featuring an ALGOL-like instrument language and enhanced set of unit generators.

In terms of software, the language of choice for high level performance programming in the late seventies, early eighties was "C", and so several systems that were implemented in C appeared, like Paul Lansky's "Cmix" program. Cmix is more a library than a computer music language, as Cmix is compiled and linked by a C compiler. Still, it has its own instrument description language, called "MINC", which is used to translate Cmix instruments into C source code.

Together with the book "Elements of Computer Music", F. Richard Moore wrote a system he called "cmusic". Cmusic can be seen as a reimplementation of Music V in C.

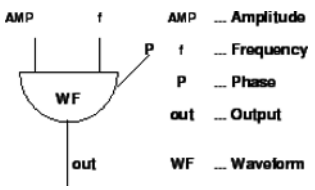
Another system widely in use today that came out of the Music V tradition is "CLM"[36], which stands for Common Lisp Music.

Although these are the more obvious successors of the Music-N languages, all modern CMS share some of the ideas of Music-N, so that we can say that its principles are generally accepted.

4. Operating Model of Music-N Languages

4.1. The Unit Generator

"There are certain strains of elements of our musical beings
that seem right be-cause they help us solve the problems that we need
to solve, such as, for instance, the unit generator" – Gareth Loy



[height=5.5cm] ugens
Figure 1.1: A unit generator diagram

```
1  INS 0 1   ;
2  OSC P5 P6 B2 F1 P30   ;
3  OSC B2 P7 B2 F2 P29   ;
4  OUT B2 B1   ;
5  END   ;
6  GEN 01100   .99 20   .99 491 0 511   ;
7  GEN 01200   .99 205 -.99 306 -.99 461 0 511   ;
8  NOT 012  1000 0.0128 6.70   ;
9  NOT 211  1000 0.0256 8.44  ;
10  TER 3   ;
```

Figure 1.2: Example Music V orchestra and score

Probably the most influential idea introduced by Mathews was the concept of the unit generator (UG). The idea of the UG is derived from analog synthesis, where modules such as oscillators, LFO (low frequency oscillators), envelope generators and filters get plugged together in order to realize a specific sound generation algorithm. The data paths through which these modules communicate with each other are driven by voltage signals. The fact that all elements deliver the same kind of signal makes the construction of new algorithms very flexible in analog synthesizers. This idea was directly mapped to the digital domain, the signals sampled and the modules implemented in software. The implementation of UGs in software is not straightforward though. Digital computation takes time, and the computation of one sample for every channel of audio at a rate of 44100 Hz was quite a task at the beginning of computer music. Even today, we still want to minimize the time spent computing, so we are always looking for a good balance between performance and quality.

Normally one looks at a unit generator as a single fundamental, non-divisible process, an implementation of a basic calculation primitive. This is used in order to implement higher level algorithms. The code of each unit generator can run for a specific time, "consuming" and "generating" audio data. After that a scheduler has to decide which unit generator has to be run next. If the time that a unit generator has run is shorter than its life span (in Music N the live span would be a note), then the UG has to remember its state for the next run.

4.2. The Score

While the unit generator modeled the analog synthesizer circuitry with its voltage levels as data, the main responsibility of the score is to handle timed events. On an analog synthesizer this corresponds to the human interaction like pressing keys and the trigger functions of the analog sequencer.

Besides the generation of lookup tables (the `GEN` entries in Figure 1.2, line 6 and 7) a score consists of timing information, instrument information and parameters for the instrument. It is a static data description language where each line specifies a record.

Generation of sound output is triggered by the `NOT` entries (Figure 1.2). The first parameter denotes time, the second one duration, the rest is passed to the instrument as parameters. Several `NOT` entries can be started at the same time or overlap.

The Music-N score language was soon found to be too limited for several tasks. First it is lacking verbosity, parameters can not be omitted, and there is no support for default values. But more limiting than that, it doesn't allow for the expression of algorithms.

In general the concept behind the score is still valid, if we look at it as the output, an interim representation, of the dataflow between a controller or a computer music algorithm and the synthesizer. A method for realtime scheduling and dispatching of the score of a CMS is described by Dannenberg[12].

5. Control Languages

Control languages are computer music languages that don't include the synthesis of sounds, but are used on a higher level for organizing and structuring how sound is handled in a CMS. The Music-N style score could be seen as a primitive control language. Nevertheless a score is only an explicit list of events with times and parameters, and as such does not qualify as a programming language, but is more of a descriptive language. Descriptive languages (such as score languages, MIDI or OSC) are not included in this survey, because they mainly deal with structure, not with algorithms. Our concern is not how to define structure but how to generate it automatically, termed generative languages by Loy et al.[23]. Music V supported a simple sys-

tem that was using a routine called `CONVT` that could translate and combine instrument parameter values.

The aforementioned MUSICOMP system by Hiller was one of the first higher level score description systems that implemented traditional harmonic rules. After the establishment of the Music-N paradigms, a logical step to take was to compute score files using algorithms written in (already existing) higher level languages, or designing languages that can express these algorithms.

One of these score languages was "SCORE"[38], generally a preprocessor for Music V score files which translated CPN (Common Practice Notation) to a score file, but this language was not able to generate a score automatically. Expanding on the idea of CPN notation, the system PLA[37] was implemented as an extension to the SAIL (Stanford Artificial Intelligence Language) programming language and Lisp, making automatic score generation very flexible.

"FORMES" is a score generating System written in VLISP, an object oriented system for high level description of musical processes, developed at IRCAM.
Other score preprocessing languages appeared, like "CScore" and "Cybil" for CSound. With the availability of realtime software synthesizers in the late 70s, forcus started to shift to realtime control languages, languages that were used to control synthesizers and had to react to realtime input from composers or players.

5.1. Implementation of Control Languages

Traditionally the work of a composer consists in writing a score. Besides the fact that computer music theoretically allows the composer to go down to the sample level and control every single aspect of music, in practice we find that at the lowest level of control, the amount of needed data is prohibitive. The unit generators represent the first layer of abstraction, and they are offering "instruments". Still writing a score for these instruments might be too tedious in practice and additionally a computer music composer wants to use the full power of the computer in order to generate music, not only automize the signal processing.

Whereas the set of algorithms for signal processing is relatively small, the set of algorithms that can be used for controlling these DSP blocks is nearly endless. This is the reason why several control languages were actually extensions of a general purpose language, making it possible to implement new algorithms and generate new structures easily.

This also means that at this level the composer's task includes actually programming or working together with a programmer.

5.2. General Purpose High Level Languages

Several systems have been implemented in general purpose languages which were not primarily designed for computer music. There are few languages which have been chosen for this task, the most important of which are probably Smalltalk and Lisp, both supposed to be representatives of two different styles of programming, namely object oriented and functional. There are only few languages in common use that only implement one of the identifiable programming paradigms. There might be as many programming paradigms as there are problems to solve. Maybe one of the future goals of a CMS is to find the programming paradigm that fits best to the problems in the domain.

Although it is not a high-level language in its modern definition, the "Formula" System[4] used a Forth Interpreter as its implementation language.

The "MODE" (Musical Object Development Environment)[30] is a system written in Smalltalk which supports structured composition and flexible graphical editing of high and low level musical objects. It includes a score description language called "SmOKe".

Another Smalltalk based System is "Kyma"[34], a system that depends on the separation of synthesis engine on a DSP and the control interface, which in Kyma's case is mainly graphical.

Currently the most widely used system based on a Smalltalk-like language is probably "SuperCollider"[29]. The SuperCollider synthesis engine can be controlled by other means too, for example Python or Q[17].

There are several Lisp systems, starting from the aforementioned CLM and including "Patchwork"[24], "OpenMusic"[26] and Roger Dannenberg's Nyquist "Nyquist"[11], a language evolved from ARCTIC, based on xlisp. Lisp itself was pretty successful and accepted by composers, it has a clear syntax, is interpreted and therefore highly interactive. The handling of lists of dynamically allocated data lends itself directly to the idea of writing a score, which then can be manipulated by the language in one environment. Lisp syntax is really easy conceptually, but in the long term it might be hard to read.

Another example of an application of functional language to computer music is the "Haskore"[11] system, written in Haskell, a modern functional language.
In general it can be said that the most important feature of the successful languages in this domain share one common principle, which is interactivity. This topic will be briefly discussed later.

6. Realtime Control Systems, Sequencers

We already mentioned the GROOVE system, which controlled analog synthesizers by generating sampled function values as control values in realtime on a digital computer. This allowed for algorithmic, although somewhat crude control of analog synthesizers and interaction by a player.

A more elaborate language for realtime control was "PLAY". "PLAY" could generate and manipulate control streams which could be individually clocked and sent to the synthesizer's synthesis modules. PLAY was a dataflow language, which means that the control it produces is constantly flowing, it didn't have the notion of instantaneous events. Several principles in music are event based though.

"4CED"[2] was a control language for the 4C synthesizer at IRCAM, and its score generating system was based on event processing. This means that instead of having constant dataflows, the control changes are triggered by events, which can in turn trigger other events or phrases of scores.
Max Mathews' "RTSKED"[28] was another event-based system, which had some notion of multitasking built in.

A descendant of RTSKED is the "Music 500"[31] system, which replaced the traditional score file with a system based on Mathews' "trigger and wait" idea.

"Flavours Band"[16] is another LISP based system for realtime control, which was built on the notion of phrase processors that can be manipulated and applied to phrases and has influenced systems like Stephen Travis Pope's "MODE".

Other early systems that can be mentioned in this domain are "MOXIE"[9] and "FMX", all reflecting the state of the art of realtime CMS at that time.

Another very interesting approach in terms of language is the ARCTIC[13] System by Roger Dannenberg. ARCTIC is a functional language with descriptive elements. The main semantic concept consists of prototypes for events. These events, once instantiated, can trigger other events or produce output functions. Time was an explicit value in ARCTIC. The event propagation was similar to that of 4CED.

Because of the constant hardware improvements, only a few control languages of that time survived the platform they were written for. In the mid-80s, Miller Puckette was working on a program to control the newly developed 4X machine at IRCAM. His program was called "MAX"[33] (after Max Mathews), and it was written for an Apple Macintosh computer.

Several coincidences led to the survival of MAX. One is certainly the upcoming market for personal computers like the Macintosh and the establishment of the MIDI protocol, which

started in the mid-80s. Max is an event based system, just as RTSKED or Flavours Band. It had one novelty, and that was its graphical representation of the language statements. Operation could be patched together just like in the representation everyone knew from the Music V orchestra explanations. Max communicated with the 4X via MIDI, which also made it useable for any other commercial synthesizer, and probably not as useful for computer music. MAX was also used as a control language for the development of the IRCAM Signal Processing Workstation (ISPW), the followup of the 4X[14].

The popularity of mainstream Synthesizers, that led to the definition of the MIDI standard in 1983, also led to a set of programs for controlling these devices. Due to the limited parameter space of the MIDI protocol, these control programs, called sequencers, were basically not more than a graphical user interface to a stripped down version of Music-N scores.

Although problematic, the separation of control and synthesis for realtime systems remained unavoidable until the mid-90s, when general purpose computers started to be able to take over the task of dedicated hardware.

6.1. Programming Models of Realtime Control Systems

Realtime control systems add an additional difficulty to the task of score generation. As scores generally control more than one voice or more than one event stream at the same time, the realtime control system has to allow for parallelism.

Expressing parallelism in a traditional programming language is awkward, as parts that take place at the same time have to be written sequentially, and therefore it is not trivial to get the synchronisation and timing between different event streams right.

Realtime control systems can not work with absolute time, such as some score generation systems do. The systems should therefore be able to schedule events in a determined future and to react to events from the outside immediately.

Reacting to events is commonly solved by callback functions, while the scheduling needs a queue where events can be scheduled and triggered at the correct times. These events can then be processed just as events from the outside (some controller or user interaction).

It is desirable for realtime control to be able to change the behaviour of the system while it is running, in order to make it possible to experiment with the system. This means that the callbacks can be rewritten and reloaded into the system.

These properties also call for an interpreted language to make realtime control processing feasible. For this reason, signal processing and control processing are separated in most systems today, since signal processing is hard to do efficiently in interpreted languages.

7. Realtime DSP Languages

Soon after computers were fast enough to do signal processing in realtime, systems for sound synthesis began proliferating. Several of the surviving synthesis programs of the post Music V era started to work in realtime, interestingly with only a few adjustments. Software sound synthesis in the style of CSound, cmusic and Cmix had their own evolution during the years of expensive and dedicated computer music workstations. The time that it took to render a CSound score to disk got smaller and smaller from year to year, until it was shorter than the actual piece, meaning that it could run in realtime.

An interesting adaptation of the Music N orchestra file for realtime interaction is the "M orchestra language"[32] of the Music 500 system, running on a special purpose array processor.

Besides the Music V style languages, another survivor of the era of signal processing on general purpose computers was Max, and specifically its successors Pure Data, jMax and MAX/MSP.

Having the whole system running on one computer facilitated several aspects of the control problem, but did not alleviate it entirely. Obviously the problem of controlling sound processing and synthesis specifically in realtime is not a problem of the synthesis/control separation, but lies deep within the algorithms themselves[20].

The 90s saw other trends in computer technology showing up, one of which is the internet. In order to be able to handle bandwidth problems, there was yet another modern incarnation of the Music V paradigm, which was called "SAOL" (Structured Audio Orchestra Language) with its score language "SASL"[35]. SAOL is an MPEG-4 standard, but it was not widely adopted, perhaps because of the lack of an efficient interpreter.

Another field that has opened with the availability of fast desktop computers are libraries and frameworks that can be used to build CMSs. They cover different areas, from simple sound processing libraries to complete cross-platform solutions. Systems that should be mentioned here are the Synthesis Toolkit (STK)[10] and the sndobj[22] library, which implement signal processing or instrument algorithms, and and the CLAM[5] framework, which offers everything from signal processing, scheduling, graphical user interface, sound hardware access up to a whole metamodel of music computation[3].

8. Current Developments

Some of the systems mentioned above are still heavily in use and evolving. Other systems are very recent, and it remains to be seen whether they will be of the same importance as some of the Systems we have seen so far.

As a quick wrap-up, I want to mention some of these systems. First, there are more Max style implementations, "Open Sound World"[1] can be considered as one of these. It tries to overcome several shortcomings of the (already 20-year-old) Max paradigms, fighting against a strong user-base of Max and Pure Data.

Other interesting additions to the Max paradigm are graphic rendering libraries. With these, the Max paradigm also attracted interest from visual artists.

A new approach on how to handle the problems in Computer music is being tried by "ChucK"[41], which introduces the concept of a "strongly timed language", referring to its sample synchronous scheduler, and the expression of "on-the-fly" programming, a technique used in order to improvise computer music, also known as live-patching, just-in-time programming and live-coding.

The "CLAM"[5] framework is on its way to setting a new standard with higher level datatypes for spectral processing and in the rather new field of music information retrieval. CLAM is also seen as a replacement for traditional scientific tools such as Matlab.

The "Marsyas"[40] system is yet another analysis/synthesis library with a special focus on music information retrieval, but lately it also handles synthesis.

Other recent systems are based on functional programing (e.g. Q-lang)[17], like "Chronic"[7] or the research system "Varese", based on the Lisp dialect scheme[18].

Java also has its CMSs in "JSyn"[8] and "JMusic"[39], and for music description the "JMSL" (Java Music Specification Language), successor of "HMSL", the Hierarchical Music Specification Language.

Several systems are less concerned with programmability than with implementing a system based on the more traditional concept of modular synthesizer/sequencer combination, like "AudioMulch"[6] or Bidule.

Also SuperCollider shows a modern, object oriented system that offers a language shell for interaction and a low latency signal processing server for number crunching.

The proliferation of computer music software has definitely changed the ways in which composers work with computers. Nevertheless, there is still an interest in CMS developments, especially for interactive art and live performances, as in most of these cases the system plays a key role both in aesthetics and in the technical effort that is needed for its realization.

9. Computer Music Systems Today

Today, computer music systems in use can be split into several classes. The biggest one is probably the commercially most successful one. It includes sequencers, general software synthesizers and systems that are based on the traditional principle of mouse interaction. These systems tend to be easy to use, but due to their lack of flexibility, they are not useful for certain tasks of computer music and interactive art.

The more interesting group comprises systems such as those described above. One could say that these systems are built on two main concepts today.

One concept is traditional programming, represented by CSound, ChucK and SuperCollider, for example, the other one is graphical programming, represented by Pd, MAX, Reaktor and several others.
Evolving are hybrid forms, like CSound instrument editors or scripting language support for graphical applications.

9.1. Traditional Programming

Traditional programming forces the user or programmer to be able to abstract the task to a high degree. When using these systems, two principles come into play. The first one is the memorization of the elements of the language. Languages can generally be defined by a fundamental set of allowed tokens or words and a syntax that is used to define allowed statements. The memorization of the tokens is normally an easy task, but nevertheless one that has to be done before being able to work with the language. This task can be made easier by a built-in list of allowable tokens and auto completion.

The syntax is the way these words can be put together in order to form correct sentences. The difficulty of general programming languages starts when trying to learn the syntax of a language. For a programmer, this is easier, because programming language syntax tends to be based on the same principle. However, computer music systems might need some less common extensions. Nevertheless, before being able to start to use a system based on traditional programming, the user has to learn the syntax, at least to a point where it admits him to express a handful of useful statements. This can be compared with learning foreign languages. (It is not by chance that the first programs people normally write in a new language are for printing, "Hello world", "hello" being the first word someone would learn in a foreign language too).

There is still a third level that has to be reached when acquiring the knowledge to be able to use a text-based computer music system, it is called the semantics. The semantics of a language is the part that gives sense to the sentences. Although one might build perfectly correct sentences (statements), the idea that they express might still make no sense.

Textual computer music systems traditionally do not offer much help for learning. Generally the user has to go through a steep learning curve in order to be able to express ideas within the systems. Furthermore, some of the ideas behind computer music systems are fairly abstract. Most of the time, textual systems offer little or no help for understanding these concepts, which makes them harder to use for some users.

The task of debugging in textual systems generally consists of printing important program-internal information on the screen and checking whether the program is doing something wrong.

The advantage of a well designed textual computer music systems is its flexibility. Especially if the system is built upon a general purpose programming language, the possibilities are endless.

9.2. Visual Programming

Some of the problems with textual languages might be alleviated by visual programming paradigms such as those offered by modular synthesizers and, in a more flexible way, by programs like Max or Pd. It is still necessary to learn the meanings of the program tokens (objects in this context). The language can provide help with the syntax problem, by allowing certain connections and refusing to connect non-compatible objects, actually performing the syntax check after each word that gets added.

The semantics of the language are difficult to master, especially if the language offers a reasonable amount of flexibility. Low flexibility systems like modular synthesizers are easy to understand and program. If high flexibility is needed, more effort has to be put into designing and programming a system.

Even with flexible systems, most of the time it is difficult to express algorithms that would be just a few lines of code in a normal textual programming language. This has led to a proliferation of custom written extensions to Max and Pd. The goal has not yet been reached of having a complete, self-contained system with a small set of principal objects that can be used to express almost every algorithm.

Probably the biggest advantage of visual programming languages is their interactivity. Pd, for example, allows changing the program while it is running, making it possible to develop, debug and design a patch at the same time. This is very appealing to users that do not come from a programming background and has probably been the key to the success of the Max paradigm. On the other hand, this same property sometimes leads to badly written (patched) programs.

9.3. The Future

Visual programming is certainly a more helpful model for program design, although visual languages are still far from being able to compete with textual languages. The best solution seems to be a hybrid system, where textual subroutines or objects can be embedded into a visual system. This would allow for flexibility, cross platform development and extensibility of the system, without a proliferation of custom written binary extensions.

For example scripting extensions exist for Pd, but they are not standard, and a good integration of scripting and visual language would probably require a slightly different design of the system in general.

Experience shows that scripting languages are generally easier to handle and support under different platforms than compiled languages. Theoretically though, there is no reason why a compiled language could not be just as easy to use, as long as the compilation is not too time-consuming.

Although existing visual extensions for computer music systems have not been discussed here, this is another interesting field, where several solutions are already in use. Combining audio and visuals in one system worked out pretty well in the MAX languages.

It is evident that there is still room for the evolution of new computer music systems, which should make computer music even more accessible and easier to handle.

Bibliography

1 A. Freed A. Chaudhary and M. Wright. An open architecture for real-time music Software. *In Proceedings of International Computer Music Conference*, San Francisco, 2000. International Computer Music Association.

2 Curtis Abbott. The 4CED program. *Computer Music Journal*, 5(1), 1981.

3 Xavier Amatriain. *An Object-Oriented Metamodel for Digital Signal Processing*. PhD thesis, Pompeu Fabra Univesity, 2004.

4 David P. Anderson and Ron Kuivila. Formula: A programming language for expressive computer music. *Computer*, 24:12-21, July 1991.

5 X. Arum, P. Amatriain. CLAM, an object-oriented framework for audio and music. In *Proceedings of 3rd International Linux Audio Conference; Karlsruhe, Germany*, 2005.

6 ROSS Bencina. Oasis rose the composition – real-time dsp with audiomulch. In *Proceedings of the Australian Computer Music Conference*, pages 10-12, Canberra, July 1998. Australian National University.

7 Eli Brandt. *Temporal Type Constructors for Computer Music Programming*. PhD thesis, Carnegie Mellon University, 2002.

8 Phil Burk. JSyn: Real-time synthesis API for Java. In *Proceedings of the Int. Computer Music Conference*, San Francisco, 1998. International Computer Music Computer Association.

[9] D. J. Collinge. MOXIE: a language for computer music performance. In *Proceedings of the International Computer Music Conference*, San Francisco, 1984. International Computer Music Association.

[10] P. Cook and G. Scavone. The synthesis toolkit (stk). In *Proceedings of the International Computer Music Conference*, San Francisco, 1999. Computer Music Association.

[11] R. Dannenberg. The implementation of Nyquist, a sound-synthesis language. *Computer Music Journal*, 21(3):71-82, 1997.

[12] Roger Dannenberg. A realtime scheduler/dispatcher. In *Proceedings of the International Computer Music Conference*, pages 239-242. Computer Music Association, September 1988.

[13] Roger B. Dannenberg. Arctic: A functional language for real-time control. In LFP '84: *Proceedings of the 1984 ACM Symposium cm LISP and functional programming*, pages 96-103, New York, NY, USA, 1984. ACM Press.

[14] M. Puckette E. Lindeman and E. Viara. The IRCAM signal processing work-station – an environment for research in real-time musical signal processing and performance. In *Euro Micro Proceedings*, 1990.

[15] M. V. Mathews et al. *The Technology of Computer Music*. MIT Press, Cambridge, Mass., 1969.

[16] Christopher Fry. Flavors band: An environment for processing musical style. *Computer Music Journal*, 8(4): 20-34, 1984.

[17] Albert Grf. Q: A functional programming language for multimedia applications. http://lac.zkm.de/2005/proceedings.shtml.

[18] Peter Hanappe. *Design and Implementation of an integrated Environment for Music Composition and Synthesis*. PhD thesis, University of Paris 6, 1999.

[19] Paul Hudak. Haskore music tutorial. In *Advanced Functional Programming*, pages 38-67, 1996.

[20] Julius O. Smith III. Viewpoints on the history of digital synthesis. In *Proceedings of the International Computer Music Conference*, pages 1-10, San Francisco, 1991. Computer Music Association.

[21] A. Leal L. Hiller and R. A. Baker. *Revised MUSICOMP manual. Tech. Rep. 13*. School of Music, Experimental Music Studio, University of Illinois, Urbana, III, 1966.

[22] Victor Lazzarini. The sound object library. *Organised Sound*, 5(l):35-49, 2000.

[23] Gareth Loy and Curtis Abbott. Programming languages for computer music synthesis, performance, and composition. *ACM Comput. Surv.*, 17(2):235-265, 1985.

[24] Laursen M. and Dithen J. Patchwork, a graphical language inpreform. In *Proceedings of the International Computer Music Conference*. International Computer Music Computer Association, 1989.

[25] Donald Maclnnis. Sound synthesis by Computer: MUSIGOL, a program written entirely in extended ALGOL. *Perspectives of New Music*, 7(1) :66-79,1968.

[26] Carlos Agon Marco. High level musical control of sound synthesis in open-music.

[27] M. V. Mathews and F. R. Moore. GROOVE program to compose, store, and edit functions of time. *Commun. ACM*, 13(12):715-721, 1970.

[28] Max Mathews and J. Pasquale. RTSKED, a scheduled performance language for the crumar general development system. In *Proceedings of the 1981 International Computer Music Conference*, page 286, San Francisco, 1981. International Computer Music Association.

[29] James McCartney. A new, flexible framework for audio and image synthesis. In *Proceedings of the International Computer Music Conference*, pages 258- 261, San Francisco, 2000. International Computer Music Association.

[30] Steven Travis Pope. The Musical Object Development Environment (MODE) – ten years of music software in Smalltalk. In *Proceedings of the 1994 International Computer Music Conference*, San Francisco, 1994. International Computer Music Computer Association.

[31] Miller Puckette. Music 500: A new real-time digital synthesis System. In *Proceedings of the International Computer Music Conference*, San Francisco, 1983. International Computer Music Association.

[32] Miller Puckette. The m orchestra language. In *Proceedings of the International Computer Music Conference*, San Francisco, 1984. International Computer Music Conference, International Computer Music Association.

[33] Miller Puckette. The Patcher. In *Proceedings of the International Computer Music Conference*, pages 420-429, San Francisco, 1988. International Computer Music Association.

[34] Carla Scaletti. Computer music, languages, Kyma, and the future. *Computer Music Journal*, 26:69pp, Winter 2002.

[35] E. Scheirer and B. Vercoe. SAOL: The mpeg-4 structured audio orchestra language. *Computer Music Journal*, 23(2):31-51, 1999.

[36] William Schottstaed and Rick Taube. Machine tongues XVII: CLM – Music V meets Common Lisp. *Computer Music Journal*, 18(2):30-37, 1994.

[37] William Schottstaedt. PLA: A composer's idea of a language. *Computer Music Journal*, 7(1):11-20, Winter 1983.

[38] Leland Smith. Score-a musician's approach to computer music. *Journal of the Audio Engineering Society*, 20:7-14, 1972.

[39] Andrew Sorensen and Andrew Brown. jMusic – Music composition in Java. <http://jmusic.ci.qut.edu.au/>.

[40] George Tzanetakis and Perry Cook. Marsyas, a framework for audio analysis. *Organised Sound*, 4(3):169-175, 1999.

[41] Ge Wang and Perry Cook. ChucK: A concurrent, on-the-fly, audio programmgin language. In *Proceedings of the International Computer Music Conference*, San Francisco, 2003. Computer Music Association.