Computer Music Languages . . . and the Real World

Mathias Fuchs

The limits of my language are the limits of my world.

—Ludwig Wittgenstein

The language of the Canadian Eskimos has more than 10 different words for 'frozen water'. There is one for ice that melted and froze again, one for ice that is extremely cold and at least eight others. If one wants to have a conversation about frozen water in a Central African dialect, the conversation will be much more difficult. There are no appropriate words.

Yet the matter is even worse when one tries to communicate musical ideas on a digital computer. Such attempts have existed since 1957 and are known as computer music languages. A short survey should point out some of the problems in that field. Special notice will be taken of the state of the art of general-purpose programming languages at that time.

Computer Music Languages

MUSIC I (Mathews, 1957) made it possible to generate sounds of a triangle waveform on a computer. The tones could be controlled in pitch, amplitude and duration. As there were no means of defining structures on a higher level, the way of 'programming' a musical piece resembled the way programming in general was done at that time. FORTRAN was only a year old; therefore, most of the programs were still written on the assembler level. For that reason, the early programs for computer music contain all the ugly ingredients of 1950s programming technique, like GOTOs, mnemonics, numeric labels, and so on.

MUSIC III (Mathews, 1960) presented the totally new concept of unit generators. The program was a simulation of electronic modules, like the ones of the Moog synthesizer, which appeared at almost the same time. There are also parallels to programming techniques that became known at that time. Modular programming tries to build up a program structure from smaller, generalized units that fit into various applications.

MUSIC IV (Mathews, 1963) was programmed because the company changed the computer. It would be hard to find musical reason behind the redesign of the language. Mathews himself admits: "MUSIC IV was simply a response to a change in the language and the computer. . . . So in essence MUSIC IV was musically no more powerful than MUSIC III" [1]. There are many cases like that, where technical necessity rather than musical need controlled the development of music languages. Gottfried Michael Koenig, another pioneer of formal music languages, was forced to rewrite Project 1 from FORTRAN to ALGOL when he moved from Cologne to the Netherlands.

Growing discontent with the working conditions at a big university computer center prompted a number of composers to design portable systems in the beginning of the 1970s. Ed Kobrin's HYBRID system for voltage-controlled oscillators, amplifiers and filters was such a system running on a PDP 11/10. Others were the EMS1 system or HYBRID 0 for Moog modules and a CA minicomputer. In order to perform in real time, these systems ran very fast command interpreters to service the musician's instructions and the hardware interfaces. The only input source was, in many cases, the alphanumeric keyboard. Therefore, one of the main design criteria was an extremely abbreviated command language. For example, a line to set the oscillator 01 to the top tenth of the available frequency range was

01;8–9;I2;K;R <return>

One does not need to point out that such a cryptic code is only transparent to the experienced user.

The intransparency and the lack of self-explanatory strength led Xenakis to the design of the machine UPIC. Xenakis can claim to have one of the few computer systems whereby an absolute beginner immediately can start to make music in an intuitive way. This goal is accomplished through graphic programming. Instead of defining functions by numbers or mathematical relations, UPIC reads in lines that are drawn onto a graphic table and controls a software synthesizer according to the line's course. As soon as the line goes up, the pitch (or amplitude, etc.) goes up.

The problem with a complex system like UPIC is that it can be run only on a specific machine—in this case, the one at CEMAMu in Paris. Many composers would never be able to go there and experiment with the system for a reasonable time. These considerations were a starting point for the CDP (Composer's Desktop Project) in Great Britain, where an inexpensive system using the generally available Atari 1040 was built. The software is an integrated package of sound processing utilities, called GROUCHO. The link between the programs is a standardized file format. As all the programs use this format, it is easy to let the system grow. Any one of the project's members can add new software at any time. The design idea of an open system and some features of the user interface have their parallels in modern operating systems and in data exchange and communication programs. One could see this design as opposed to the totally defined languages, like PL/1. The open systems have their main domain in the home computer and PC field. Systems that resemble GROUCHO in some respects are the Californian

Mathias Fuchs, Elsa Beskowsgata 21, S-12666, Hägersten, Sweden.

Received 25 April 1988.

Pergamon Press plc. Printed in Great Britain.
0024-094X/88 $3.00+0.00

Table 1. Music Languages

Music Language	Author	Year of Implementation	Implementation Language	User Interface/Hardware	Representation of Objects
MUSIC I	Max Mathews	1957	Assembler	punched cards/big mainframe	numbers
MUSIC II	Max Mathews	1958			
MUSIC III	Max Mathews	1960			unit generators, simulation of electronic circuits
MUSIC IV	Max Mathews	1963	FORTRAN		
MUSIC V	Max Mathews	1968			
Project 1	Gottfried M. Koenig	1964	FORTRAN rewritten in ALGOL	command lines	numbers, strictly formatted (e.g. 4.783), lists
Project 2	Gottfried M. Koenig	1965	ALGOL 60	subprogram calls/multiuser system	numbers, lists
SCORE	Leland Smith	1972			
GROOVE	Max Mathews, Moore	1970		graphic input	
HYBRID IV	Edward Kobrin	1976	PDP Assembler	command lines/ mini comp. and piano keyboard	symbols for synth. units
UPIC	Iannis Xenakis, CEMAMu	1970s		dialogue via graphics tablet, CRT, tableau des fonctions	graphs
CHANT	Xavier Rodet, Gerald Bennett, Conrad Cummings, Yves Potard	1980	FORTRAN	parameters in dialogue	functions
BADA	Michael Hinton, EMS Stockholm	1970s	FORTRAN		
CARL	Stanford				
GROUCHO	Composer's Desktop Project, A. Bentley and others	1986	C	program calls/personal computer	

CARL (which actually served as a model for GROUCHO) and the Stockholm BADA package (see Table 1).

PROPOSAL FOR CLASSIFICATION

Much of the systematic work on computer music [2] divides the object of investigation into two classes:
(i) sound generating programs and
(ii) programs for score generation.

The first ones are often called instrument definitions. Mathews argues that this hierarchy (in his case the categories are instrument, score, and performance or interpretation) emerges naturally from our concept of conventional music. For two reasons I do not want to keep this idea.

First, electronic music has ever attempted to cross the borders between the realm of 'sound' and 'compositorial structure' and has understood them as inseparable from each other. How can a systematic theory then hold up this difference? Second, since music languages contain certain features of the general purpose languages that are available at the time of their conception, I suggest a classification that takes into account the features of the 'unmusical' computer languages.

CLASSIFICATION FOR PROGRAMMING LANGUAGES

There are at least four major families of programming languages, which differ in how they approach a programming problem (see Table 2).

(i) The first one could be called the systematic approach. ALGOL, Pascal, C and ADA belong to this family. These languages are characterized by total regulation by means of syntactic and semantic rules for any statement inside the language. Easy things are often complicated to express. Classes of objects are not permeable.

(ii) The second family was born out of the frustration caused by the first. BASIC was such a language, whereby simple tasks could be achieved in an easy way. APL is another example that shows how efficiently and briefly one can code as long as the language is flexible enough. One main aim in the design of these languages is efficiency.

(iii) The third family is fine for people who cannot read or write but still

Table 2. General-Purpose Programming Languages

General-Purpose Programming Language	First Published	Author(s)	Class
FORTRAN	1956		(i) systematic, algorithmic
ALGOL 60	1960	Backus et al.	
PASCAL	1971	Nickolaus Wirth, TH Zurich	
C			
ADA	1979	Jean Ichbiah, Honeywell Bull, Department of Defense	
BASIC			(ii) efficient
APL	1960s	Kenneth E. Iverson, IBM	
LOGO			(iii) graphic
SMALLTALK	early 1970s	Alan Kay, Xerox Palo Alto	
Framework	1984	Ashton Tate	(iv) integrated, open
Framework II with FRED programming language	1986		
OpenAccess			
1-2-3		Lotus Development Corporation	
Symphony	1985		

Table 3. Music Languages

Music Language	First Published	Author	Class
MUSIC I, II, III, IV, V	1957–1968	M. Mathews	(i) systematic, algorithmic
Project 1, 2	1964, 1965	G. M. Koenig	
HYBRID IV	1976	E. Kobrin	(ii) efficient
UPIC	1970s	I. Xenakis	(iii) graphic
GROUCHO	1986	CDP	(iv) integrated

want to be programmers. It is the family of graphic languages. LOGO is such a language, but the ideology reaches out into many modern operating systems, especially in the personal computer sector.

(iv) It was not long ago that the PC market was conquered by a type of software that became known as integrated packages. These programs, though not general-purpose programming languages in the original meaning of the word, come close to them when one regards standard applications for everyday usage. Programs like Framework, OpenAccess, Lotus Symphony and others offer many convenient functions to make programming easier. Usually, they contain some graphic utilities, a database, spreadsheet functions, a word processor and communication programs. A characteristic feature is their open design, which allows linkage to all kinds of programs. They are less hermetic, less concise, easier to use and easier to learn than the systematic high-level programming languages.

The classification proposed here reflects the way of handling the objects of the language. There are, of course, languages that fit into more than one group. SMALLTALK is one; though the graphic facilities are an important part of the language, one could place it as well in class (i).

CLASSIFICATION FOR MUSIC LANGUAGES

If one applies the same classification scheme to computer music languages, one notices the degree to which design criteria for musical languages rely on design trends in the commercial field (see Table 3). It is certainly not by chance that the birth of the most interesting music packages, CARL and GROUCHO (Stanford and York, respectively), coincides with the birth of the Symphony/Framework class.

CONCLUSIONS

Language is a virus.

—William S. Burroughs

One of the myths computer musicians are living with is the idea that they can use the computer as a neutral tool that serves them in realizing their compositional ideas without influencing them. The opposite is true.

Musical output often seems closer to the computer system used than to the composer's ideas (if such were present). As a consequence, record covers and concert reviews are more often based on the technological background of the pieces than on aesthetics. The tool is not a neutral means of achieving an abstract aim; rather it should be regarded as an important factor in the process of musical creativity. This is no shame, neither is it new to the musical universe. It makes a difference whether one writes for trombone or for guitar. The impact of a computer music language on the process of composing therefore should be seen in the context of a given state of the art in programming techniques. Computer music languages have to do with sounds and with musical structures. But they also

have to do with which software products are commercially available.

The German composer Herbert Eimert was angry once when somebody associated electronic music with electricity. He said, "Electronic music has more in common with serialism, than with the electric vacuum cleaner".

Apart from the fact that the concept of the vacuum cleaner seems to have a more lasting life than that of serialism, we should not hesitate to observe the importance of the hardware behind artistic creation.

References and Notes

1. *Computer Music Journal* **4**, No. 4 (1978).

2. For further information the reader is directed to the following works: William Buxton, *Design Issues in the Foundation of a Computer Based Tool for Music Composition* (Ontario, 1978); Lejaren Hiller, *Computermusik und Informationstheorie* (Mainz, 1963); Michael Longton, *Priorities in the Design of a Microprocessor-Based Music Program* (Victoria, 1981); Max Mathews, *The Technology of Computer Music* (Cambridge, 1969).

Further Reading

Ed Kobrin, *Computer in Performance* (Berlin, 1973).

Gottfried Michael Koenig, Computer Composition (Sonological Reports, Utrecht).

Barry Truax, POD 4,5 and 6 (Sonological Reports, Utrecht).

In addition, the reader should consult the diverse manuals and printouts of SCORE, BADA, UPIC, CHANT, Hybrid 0.