Musical Software: Descriptions and Abstractions of Sound Generation and Mixing

Marc Battier, Patrick Greussay, Jacques Arveiller, Colere Christian, Dalmasso Gilbert, Englert Giuseppe, Roncin Didier

Patrick Greussay
Jacques Arveiller
Marc Battier
Chris Colere
Gilbert Dalmasso
Giuseppe Englert
Didier Roncin

Groupe Art et Informatique Vincennes
Université Paris VIII
Paris, France

Musical Software: Descriptions and Abstractions of Sound Generation and Mixing

1. Introduction

Software research for musical applications at the Groupe Art et Informatique Vincennes (GAIV) has been devoted almost exclusively to the interaction between the software system and the interpreter/composer (IC). We are especially interested in the real-time aspects of this interaction. Naturally we have been led to consider this interaction in terms of:

1. Process
2. Hierarchies of and interactions between processes
3. Data flows
4. Paths for and operators on data flows

We feel that the IC itself represents a process which encounters, interacts with, and controls other processes. These processes can be other ICs, or system processes defined in the software.

This type of approach to musical software is related to other current research, especially that dealing with data flow languages (Dennis 1974), communication among parallel processes (Atkinson and Hewitt 1976; Hewitt and Baker 1977), and the description and formalization of coroutine networks (Kahn and McQueen 1977). We will describe new kinds of operators and configurations of pipeline stages that have been implemented at GAIV as a result of the musical and interactive character of the processes we have studied. Our implementations have been based on the Vincennes version of LISP, VLISP (Chailloux 1975) and the low-level language INTELGREU.

The set of process descriptions we will present has not been based on theoretical presuppositions. Rather, it has been formulated on the basis of experience gathered from the use of our systems in a performance setting, which has allowed us to gradually correct and perfect these systems, as well as from analysis of musical works from the past in terms of data flow and data base (Greussay 1973). Performance experience seems indispensable to us for the verification of software tools, abstract or real. Indeed, contrary to the studio situation, we are faced *immediately* with the consequences of our decisions: from the outset, unrealistic or unfeasible viewpoints must be eliminated. Furthermore, we can experiment with the actual representation of musical processes employed by professional musicians, following the lead of experiments on the representation of musical processes in children carried out in the LOGO laboratory at M.I.T. (Bamberger 1976). Analysis by computer of musical works from the past allows us to verify our concepts in compositions. One of the theories supported by GAIV is that a successful composition is, in a certain sense, an improvisation (Dalmasso 1980). In this way we are trying to think, in computer science terms, of the processes of (score) reading, foreshadowing of later events in a piece (or the lack of it), improvisation, planning, and composition.

Finally, we should add that we are not trying to establish a theory of musical/instrumental processes. To be sure, we shall, without exception, use a schematic representation to describe data and pro-

Computer Music Journal, Vol. 4, No. 3, Fall 1980,
0148-9267/80/020040-08 $04.00/0

Fig. 1. Total dependence (top) between an interpreter/composer (IC) and one or more system processes (SP), and (bottom) between system processes and sound generation devices (SG).

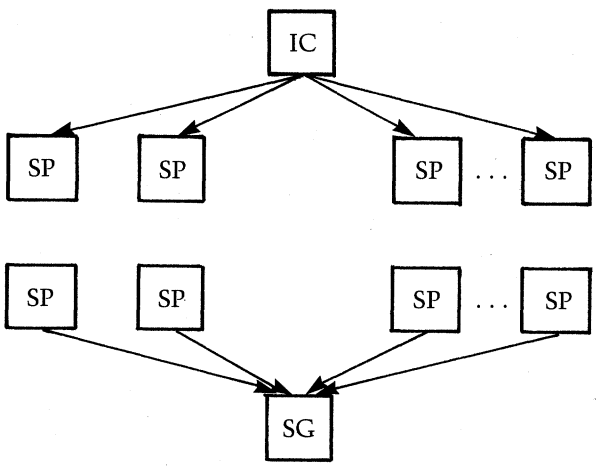
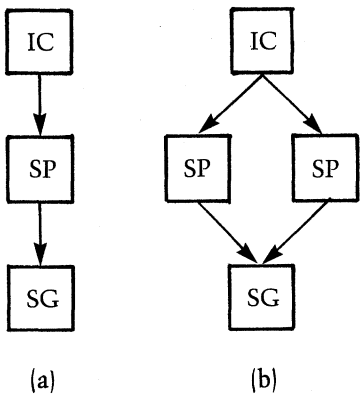


Fig. 2. Two possible patterns of total dependence: (a) the noninteractive case; (b) the interactive case.



cesses. But the formal descriptions we will discuss represent nothing more than abstractions of software tools that are being planned or have actually been completed. Our only purpose, in fact, is to make it easier for musicians at GAIV to approach a computer science that deals with the interactive use of computers.

Thus in the following brief presentation we prefer to emphasize the fundamental concepts on which our software is based, rather than specific details of implementation. The latter are described in detail in the internal reports of the University of Paris VIII and in our journal *Artinfo/Musinfo*.

2. Abstractions of Dependency Relations in Sound Synthesis

We will first describe certain kinds of abstract dependency relations that can occur between (1) an IC; (2) one or more system processes (SP), for example, processes for choosing overall frequency, density, or amplitude; and (3) one or more devices for the generation of sound (SG).

2.1. Total Dependency

This kind of relationship is a characteristic feature of noninteractive musical software. As can be seen in Fig. 1, all of the processes for the entire composition are determined in advance. The control of SP by IC, and of SG by SP, is total; no problems with scheduling or synchronization can occur. The SPs are conceptually independent and cannot communicate among themselves. Whatever the number (necessarily finite) of SPs, the potential relationships can be reduced to two simple patterns. Figure 2(a) shows the noninteractive case (e.g., for a composition realized in the studio). For total dependence in the interactive case (Fig. 2[b]), the IC control should operate on no less than two SPs because at least two decisions are necessary. This kind of control was used in the KRWTH system (Chailloux 1975). Note that for the sake of simplicity we show only one SG. Actually, several SGs are connected by the abstract mixing operators, which we will describe later (Section 3).

2.2. Levels of Hierarchical Relationships (IC SP)

We have here the case where the layer of SPs at level $i \epsilon [1, n]$ can control or create the layer of SPs at level $i+1$ (cf. Fig. 3). Examples of this would include software for aleatoric synthesis, and pattern generators (cf. Fig. 4). We see that the dependency is always hierarchical, that is, it operates from layer to layer. Furthermore, the SPs of a single layer cannot interact with one another. Finally, the number of layers is determined by the program in a fixed and definitive way, with only one IC at the top level. Just as in Section 2.1, hierarchical relationships are not limited to interactive musical situa-

Fig. 3. Hierarchical relationships among various levels of SPs.

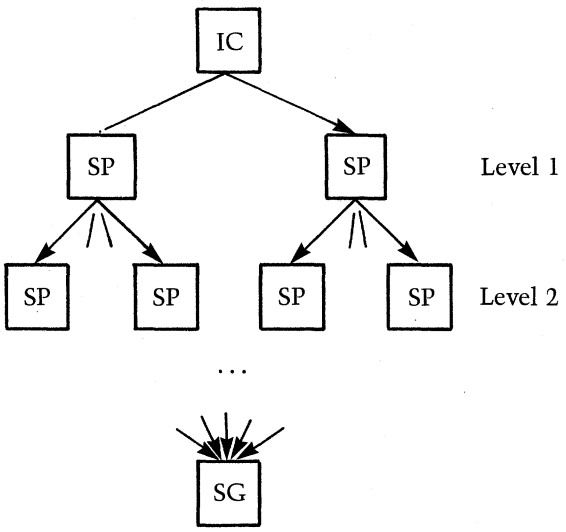


Fig. 4. An elementary example of a pattern generator in VLISP. All possible recursive compositions can be created using these two elementary patterns.

```
(DE GENPAT (PAT A B C D E F)
    (LET ((PAT PAT)) (COND
          ((NULL PAT) (NIL)
          ((ATOM PAT)
           (APPEND (EVAL PAT) NIL))
          ((NUMBP (CAR PAT))
           (GENCOP (NEXTL PAT) (SELF PAT)))
          ((EQ (CAR PAT) ',)
           (GENCOP (EVAL (CADR PAT))
                   (SELF (CDDR PAT))))
           (T (APPEND
                    (SELF (NEXTL PAT))
                    (SELF PAT))))))
(DE GENCOP (N SQ)
    (LET ((N N))
         (IF (= N 0) NIL
             (APPEND SQ (SELF (SUB1 N))))))
(GENPAT '(A B A) sq1 sq2)      sq1 sq2 sq1
(GENPAT '(n . A) sq)      sq sq . . . sq
← n TIMES →
```

tions and can be summarized in the two patterns shown in Fig. 5.

2.3. Mutually Dependent (Horizontal) Relationships

There is also the possibility of communication and dependency within a single layer. The three varieties of *horizontal* relationships, shown in Fig. 6, were used in the program RE COSA MATERIALE (Battier 1977).

For lack of space, we cannot give specific, detailed examples of each type of dependency. We should at least point out, however, that formally speaking, these relationships among a fixed number of SPs at a given level constitute a network of finite automata. If the SPs are ICs, we have the kind of interdependence characteristic of *group improvisation* (Dalmasso 1980). Jacques Arveiller used this kind of organization in the computer program for *Paire-Lacs*, created at Tours in April 1976. In this kind of organization we must deal with the question of synchronization and scheduling due to the limited number of available resources (Battier 1977). In addition, if the upper layer is constructed of more than one IC, then the system must use special IC-SP communication terminals featuring a handler for various interrupts and a keyboard, along with an inter-IC communication system consisting of a common clock, display screens, and light signals. Having gathered some experience at a concert in Frankfurt (in April 1977) with a special color TV terminal constructed at the University of Paris VIII by Louis Audoire (1976), GAIV is now planning to generalize this kind of inter-IC communication.

2.4. Recursive Dependency

In this kind of relationship (Kahn and McQueen 1977) we are confronted for the first time with a layer that exists separately from the SP (Fig. 8). As a matter of fact, an SP in layer i can depend on several distinct SPs in the layer just above it $(i-1)$.

Furthermore, the number of SPs is no longer fixed in advance; a recursive SP call can produce a new SP network. Since the capacity of memory and process resources is finite, we must also introduce a new kind of process, known as *reclaiming*, which is responsible for "garbage collection." This extremely

Fig. 5. Hierarchical relationships. The noninteractive case (a) results in a single and specific decision. In the interactive case (b) at least two IC decisions are necessary (Menard 1974).

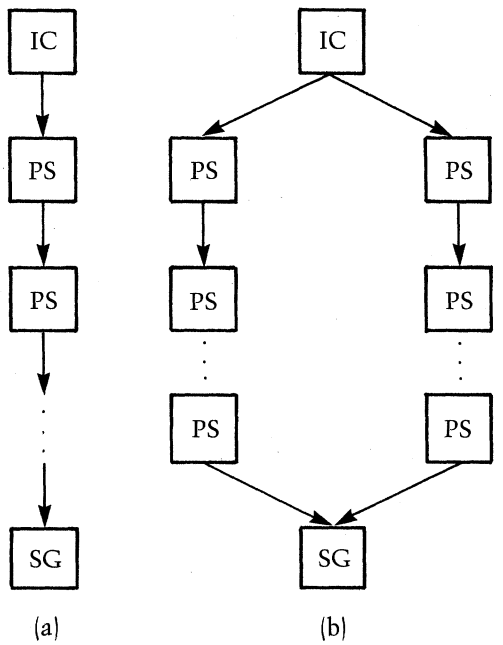
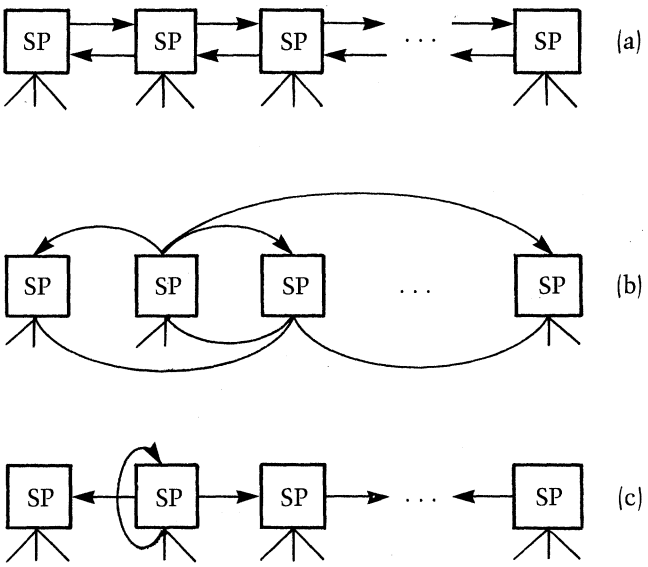


Fig. 6. Dependency through a "closest neighbor" bilateral relationship (a). Hierarchical dependency (b). Auto-dependency (c).



interesting organization introduces strict limitations on the creation of music in real-time or in a performance situation. We must use techniques of "incremental garbage collection" to allocate a new SP within some given length of time, regardless of the number of inactive SPs to be recovered (Baker 1977).

Finally, we should point out that this kind of relationship between SPs has been found through the analysis of the structure of improvisation (Dalmasso 1980), as well as analysis of compositions that are apparently quite removed from our interactive approach (Greussay 1973).

2.5. Independence of Layers: Total Parallelism

At Vincennes, lacking hardware multiprocessors, we have only been able to experiment using software simulation (Greussay 1978). Although the possibilities are quite complex, we will discuss the two simplest organizations, the AND network and the OR network (Fig. 9). In these two cases we see first that the logical distinction between SP and IC is abolished, which necessitates the construction of completely new link terminals. Furthermore, the relationships among the SPs are no longer fixed but can be modified within the network through the *propagation of constraints* (as shown in Fig. 10). This kind of relationship introduces the concepts of distributed monitors, perturbation monitors, and fracture monitors, which are outlined in Englert's article (1977).

We can consider this last kind of organization as the most interactive, compatible with current software/hardware tools. It introduces extremely difficult and interesting problems of process maintenance. At the same time it is in all probability the model closest to what we know about the interactions among human musicians.

At this juncture, we should point out that the interactive aspect of computer composition is characterized by the control of perturbation propagation. Our next step will be to experiment actively with more complicated relationships than AND networks and OR networks. Louis Audoire and Didier Roncin have constructed specialized inter-IC terminals adapted to this kind of organization. These include a terminal with a color screen and a VLISP system on an LSI-11.

Fig. 7. Interconnection of ICs.

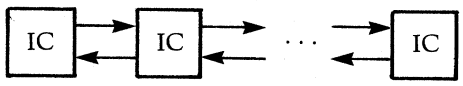


Fig. 8. Recursive dependencies (see text for explanation).

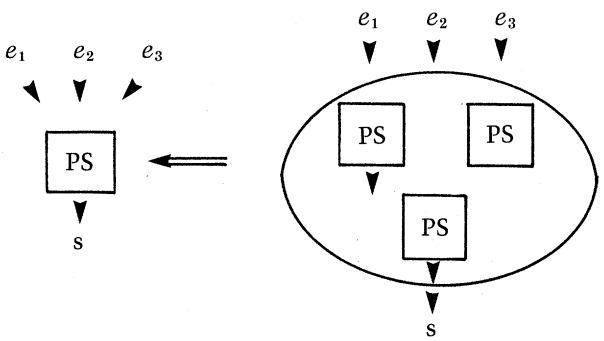
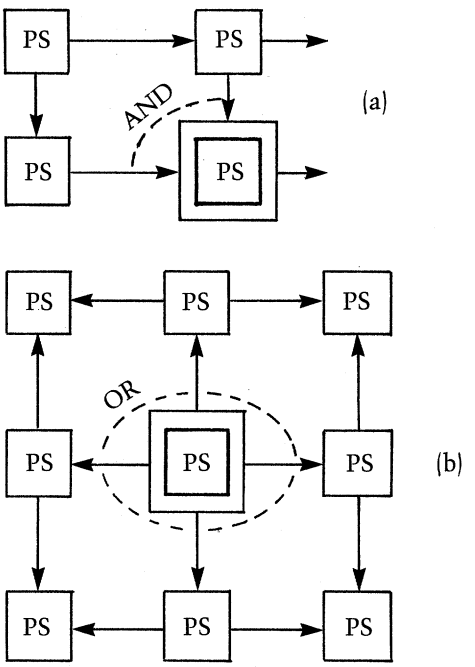


Fig. 9. AND network (a), in which the output of the SP in the lower right-hand corner depends on the simultaneity of interaction with the adjacent SPs. OR network (b), in which the adjacent SPs depend alternately *(the simultaneity then being a special case) on the SP in the middle.*



3. Abstractions in Mixing

All of the GAIV programs incorporate the features discussed in this section. The following description deals essentially with the relationships between SGs (more specifically, the program derived from the montage *Echologique*). The same description can likewise apply to the relationships among SPs, but here (as opposed to the relationships described in Section 2) we are limited to a fixed number of SGs.

Likewise in contrast with the inter-IC or inter-SP relationships, which are dependency relations, the abstract relationships in mixing, which involve the SGs, operate on streams of sequential data representing the characteristics of sound. All of the abstractions relevant to mixing can be summarized in the diagram shown in Fig. 11.

3.1. Data Flow Operators

The two forms of data-flow operators are shown in Fig. 12. These are, of course, the usual relationships found in oscillator networks.

3.2. Data Flow Pipeline Stages

The introduction of a pipeline stage allows us to use SGs in a quasi-parallel fashion. A SG can produce a datum and place it into a pipeline stage without necessarily taking into account the SGs that are going to use the datum. Introduction of a pipeline stage thus allows for the data synchronization of processes. The data-flow pipeline stages are known as *cells*. The Bi-Sequencer program (Greussay 1978) allows the use of an arbitrary number of independent processes synchronized through pipeline stages.

A most common example of data-flow synchronization is the *time loop*, shown in Fig. 13.

We distinguish four configurations, ranked according to increasing complexity and shown in Fig. 14. If no pipeline stage is present, the data flow passes immediately from one SG to the next. With a single intervening pipeline stage (Fig. 14[b]), SG_B takes its input datum from cell *c* where the datum of SG_A is deposited. No time sequence is fixed *a priori* between SG_A and SG_B. As a matter of fact,

Fig. 10. Example of a potential change in a totally parallel configuration.

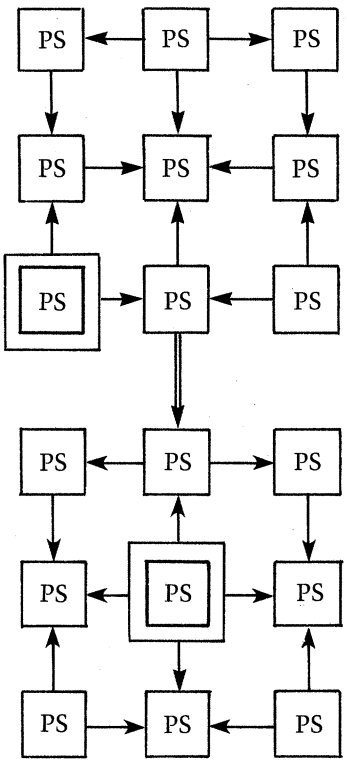


Fig. 11. Abstraction of mixing.

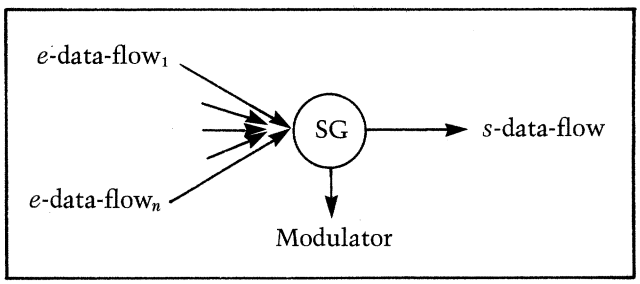
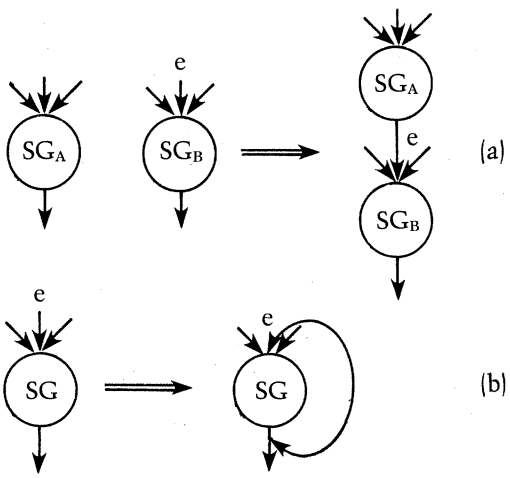


Fig. 12. Data-flow operators: (a) interconnection; (b) feedback.



SG_A can produce its datum without SG_B being active, or SG_B can be active and SG_A inactive. A request for data by SG_B can also be used to activate SG_A.

If an SG is logically decomposed into n sub-SGs, n cells are necessary, as shown in Fig. 14(c). Each of the SG_As can independently process an output datum. According to its needs, SG_B will take its input datum from cells c_1, c_2, c_3, and/or c_4.

If a *queue* is placed between the two SGs, SG_A can theoretically produce an infinitely long sequence of data, which would fill an infinite number of cells. According to its needs, SG_B uses the elements of this sequence in first-in, first-out (FIFO) order. In practice, since the total number of cells is limited, SG_A must stop producing data and temporarily deactivate itself when the limit of the number of available cells is reached. SG_A will be reactivated when SG_B, by taking an input datum, makes cells available.

4. Conclusion

At the present time we are interested in developing software adapted to networks of microprocessors. For us the elimination of the simulation of these networks in software has become a primary goal, not only because simulation is obviously less efficient but also because we want to make use of our special visual terminals. Perhaps a bit paradoxically, we feel that further progress in the development of a computer science dealing with the interactive, musical use of computers rests on the development of these new visual terminals in all cases where several ICs must interact during the performance of a single piece in real-time.

Fig. 13. Data stream synchronization. The output datum s_{i+n} derives from the input datum l_i, s_{i+n-1} derives from l_{i-1}, and so on.

Fig. 14. Four ways of configuring SGs with pipeline stages. No pipeline stage (a). A single pipeline stage (b). Multiple stages (c). Queue (d).

Fig. 13.

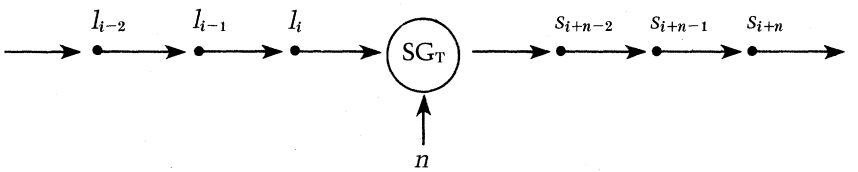
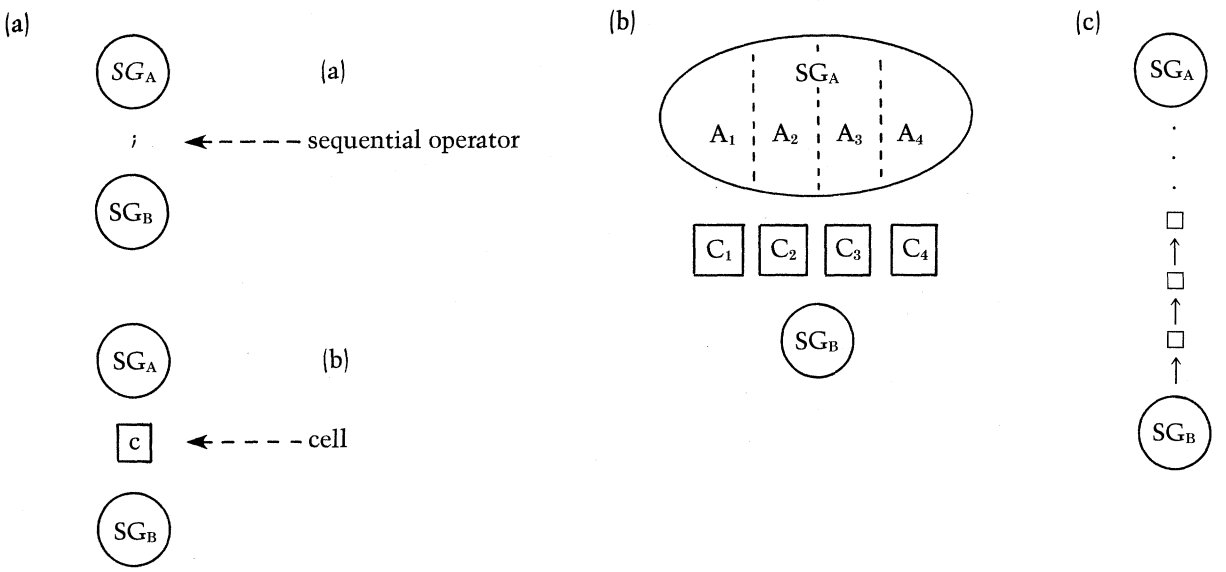


Fig. 14.



Acknowledgment

The editors of *Computer Music Journal* would like to express their appreciation to James A. Moorer for his help in preparing this manuscript.

References

Atkinson, R., and Hewitt, C. 1976. "Parallelism and Synchronization in Actor Systems." Draft. Cambridge, Massachusetts: M.I.T. Artificial Intelligence Laboratory.

Audoire, L. 1976. *Colorix: un périphérique de visualisation couleur*. Mémoire de Maîtrise d'Informatique. Paris: Université Paris VIII.

Baker, H. G., Jr. 1977. "List Processing in Real Time on a Serial Computer." Working Paper No. 139. Cambridge, Massachusetts: M.I.T. Artificial Intelligence Laboratory.

Bamberger, J. 1976. "Children's Representations of Pitch Relations." Memo No. 43. Cambridge, Massachusetts: M.I.T. LOGO.

Battier, M. 1977. "RE COSA MATERIALE et le programme compositionnel ICOSA." *Artinfo/Musinfo* 27:21–45.

Chailloux, J. 1975. "KRWTH." *Artinfo/Musinfo* 20:1–16.

———. 1976. *UER Informatique*. RT 17-76a. Paris: Université Paris VIII, Vincennes.

Dalmasso, G. 1980. "Musique et improvisation: procedures de descriptions symboliques d'environments dynamiques." Ph.D. disseration, Université Paris VIII.

Dennis, J. B. 1974. "First Version of a Data-Flow Procedure Language." In *Colloque sur la programmation*, ed. B. Robinet. New York: Springer, pp. 362–376.

Englert, G. G. 1977. "Fragola." *Artinfo/Musinfo* 22:1–8.

Greussay, P. 1973. "Descriptions de procédures de lecture." *Artinfo/Musinfo* 19:1–30.

———. 1978. "Modélisation de résaux de multi-séquenceurs." *Artinfo/Musinfo*.
Hewitt, C., and Baker, G. 1977. "Laws for Communicating Parallel Processes." Working Paper No. 134A. Cambridge, Massachusetts: M.I.T. Artificial Intelligence Laboratory.
Kahn, G., and McQueen, C. B. 1977. "Coroutines and Networks of Parallel Processes." *Proc. IFIP 1977*. Amsterdam: North Holland, pp. 993–997.
Menard, P. 1974. "Présentation du programme AUTOMUSE." *Artinfo/Musinfo* 15:13–24.