



**CONSIGLIO NAZIONALE
DELLE RICERCHE**

Goffredo Haus [^], Alberto Sametti [^]

**coresynth: A System for the Synthesis of Music
cores Based on Petri Nets and a Music Algebra**

N. 7/78

7/91

**PROGETTO FINALIZZATO
SISTEMI INFORMATICI E CALCOLO PARALLELO**

SOTTOPROGETTO 7

SISTEMI DI SUPPORTO AL LAVORO INTELLETTUALE

**PROGETTO FINALIZZATO
SISTEMI INFORMATICI E CALCOLO PARALLELO**

SOTTOPROGETTO 7
Sistemi di Supporto al Lavoro Intellettuale
Coordinatore Cesare Maioli

Goffredo Haus [^], Alberto Sametti [^]

**Scoresynth: A System for the Synthesis of Music
Scores Based on Petri Nets and a Music Algebra**

N. 7/78

7/91

Reprint

[^] LIM - Laboratorio di Informatica Musicale
Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano.



Scoresynth: A System for the Synthesis of Music Scores Based on Petri Nets and a Music Algebra

Goffredo Haus and Alberto Sametti

Laboratorio di Informatica Musicale, Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, via Moretto da Brescia, 9, I-20133 Milano, Italy

Music structures can be described, processed, and synthesized using a more flexible kind of representation than the staff. In this article, we show how. In fact, we have chosen and organized symbols depending on instrumental needs within common music notations. The level of representation within scores is more detailed than that of music composition and more abstract than that of timbre modeling.

The new kind of representation we propose makes up a conceptual music framework with as many different levels of abstraction as the musician needs. It allows us to explicitly describe and process what we call *music objects* (both

traditional and nontraditional music objects).

A music object means anything that could have a musical meaning and that we think as an entity — either simple or complex, abstract or detailed — with a name and some relationship with other music objects. We can describe music objects at various abstraction levels within a hierarchical context — for example, the structural level, the score level, and the timbre level.

Common music notation is characterized by many different languages (one or more languages for each level of representation). Our research is devoted to the definition of just one language, suitable for every level of representation.

To identify the most suitable description tool, we sought a formal tool that

- requires few symbols,
- has a graphical form of notation,
- allows hierarchical description,
- allows the description of music objects processing,
- allows time description,
- allows deterministic/nondeterministic models,
- allows macro definitions for common structures, and
- shows the musician the score synthesized as the model executes.

Score modeling. We have been using Petri nets (PNs)¹ as the basic tool for music description and processing since 1980. In our research, we have developed certain programs to edit and execute PNs for both music and general-purpose applications.² The most recent program we developed is called Scoresynth. In it, we use PNs to describe, process, and synthesize music scores.

While we assign music objects (MOs) to place nodes for describing music information, we assign causal relationships and transformation rules to PN structures and PN parameters (marking, numeric labels, algorithms, etc.).

MOs can be processed by modifying the superimposition and juxtaposition laws within PN structure. On the other hand, PN parameters allow us to create instances of MOs that modify themselves according to the behavior of PN models during execution. Furthermore, we can represent the same musical information by PNs at a lower or higher level of abstraction by using suitable alternative modeling approaches.

Indeed, a PN model with a particular initial state (that is, initial marking) can represent a family of scores; a particular execution of the model synthesizes a specific score. By modifying the initial marking of the model, we can change the family of scores that can be produced by model executions. A special situation exists when we have a fully deterministic model. In such cases, we can produce one only score (given a particular initial marking).

The Scoresynth system is made up of

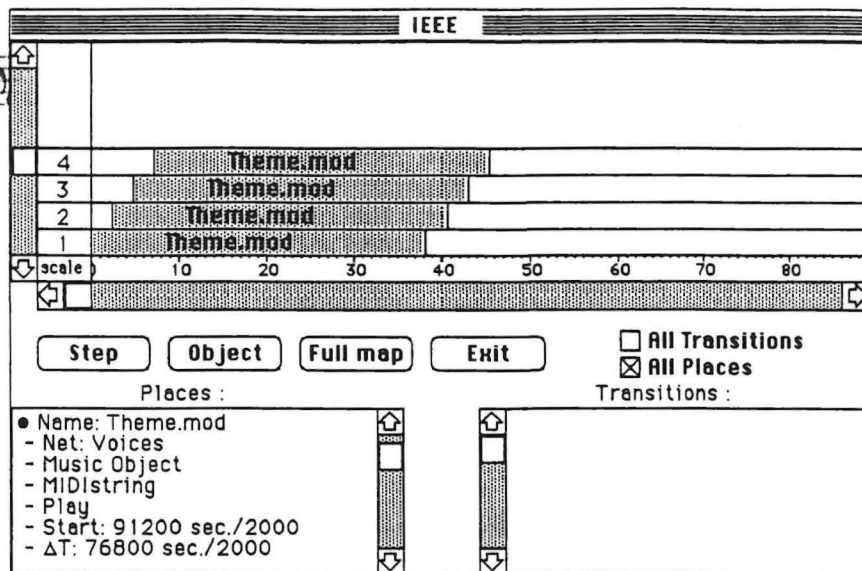


Figure 1. Music object map dialogue with the two windows of Petri net node attributes.



Figure 2. The score of the "Are You Sleeping" theme stored in the Theme file.

the editor and the *executer* of the models.

The editor allows the description of hierarchical, timed, concurrent, deterministic/nondeterministic, and weighted PN models. Both places and transitions are used as the "morphism" nodes to implement hierarchy. Common PN structures can be described as macros. An interactive graphic user interface is provided to build PNs by an iconic representation.

The executer performs the models, synthesizes MIDI (Musical Instrument Digital Interface) scores, and shows a graphic map of the MOs as they are generated by the model execution (see Figure 1). Looking at the map, the musician can interact with the model, stop the execution as soon as a bug is found in the model, and proceed to the editor environment to make modifications. This can be done simply by executing the model and looking at a graphic window that shows — on a time-MIDI_channel

plane — the map of MOs as they are appended in the score and any information requested by the musician about the genesis of MOs.

Music objects. As stated previously, the meaning of MOs is strictly related to place nodes. To us, an MO encompasses not only any sequence of notes but also something more general that is not exclusively linked to listening and not connected to the idea of process because it could last zero seconds. Obviously, the definition of an MO is partly affected by the music code standard we use (MIDI) and must be kept within the bounds of this standard (see Figure 2 as an MO example). All MIDI messages, even if not directly concerned with a note, can be put in an MO.

MOs can be coded according to three syntaxes:

- a coding language that is specific in Scoresynth,

PROJECT OVERVIEWS

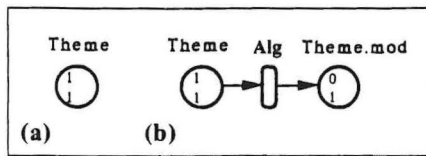


Figure 3. A place with an associated music object (a); a simple Petri net with an algorithm (b).

- an alphanumeric MIDI coding language, and
- a coding format that implements the Standard MIDI Files 1.0 format (both type 0 and type 1).

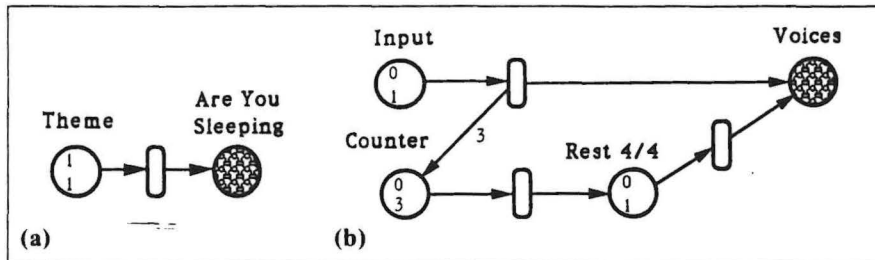


Figure 4. The IEEE net (a); the "Are You Sleeping" net (b).

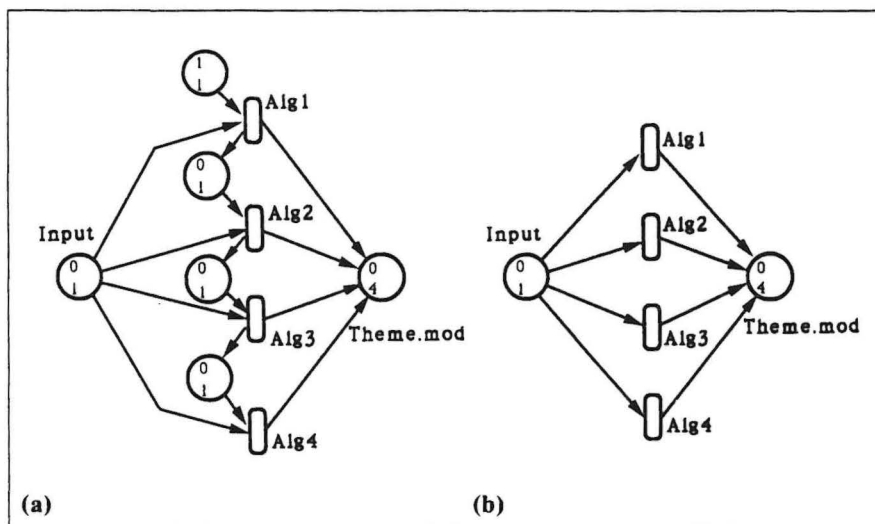


Figure 5. The Voices net: (a) deterministic version; (b) nondeterministic version.

Alg1: [C: 1, \$, [Theme, 1], 1 {...all on MIDI channel 1; use "Theme" MO}
M:1, \$, 2} {...repeat 2 times}

Alg2: [C: 1, \$, [Theme, 1], 2 {...all on MIDI channel 2; use "Theme" MO}
M:1, \$, 2} {...repeat 2 times}
P[C]: 1, \$, ? + 2} {...2 degrees transposition, C major tonality}

Alg3: [C: 1, \$, [Theme, 1], 3 {...all on MIDI channel 3; use "Theme" MO}
D: 1, \$, ? / 2} {...halve durations}
M:1, \$, 4} {...repeat 4 times}

Alg4: [C: 1, \$, [Theme, 1], 4 {...all on MIDI channel 4; use "Theme" MO}
I: 1, \$} {...retrograde}
M:1, \$, 2} {...repeat 2 times}

Figure 6. The parameter list for invoking the Voices macro Petri net.

The program uses the first format internally. The second format is based on the availability of editing tools (such as common word processors). The third format has been implemented for file import/export. MOs can be defined either by a common text editor or by a MIDI controller (a keyboard, guitar, microphone, etc.). The MOs can be stored in separate files (one file for each object) or within the PN file to which the associated place belongs.

Figure 3a shows the simplest net to perform an MO. We joined the Theme place, which has been previously stored in a file according to any of the three allowed formats, to the Theme MO.

The upper attribute of the place defines the number of tokens available, while the lower attribute specifies the maximum capacity of tokens allowed for that place.

The music algebra. The main idea of Scoresynth is to make it possible to transform both the parameters of sound (pitch, timbre, intensity, and duration) and the order in which notes appear, using algorithms. Algorithms join with transitions. If a transition has an associated algorithm, Scoresynth merges all the MOs in the entrance before making the transition fire. When the transition fires, it takes a token from each input place, applies the transformation to the single MO obtained, puts the token into all the output places, and places the outcoming MO into every output place allowed for MOs. The transformation can only be applied to note events. All the other MIDI commands are filtered and lost.

If Theme stands for the MO on which the algorithm is processed at the input, the algorithm can be applied to the whole Theme or can be limited to one of its subsequences, as designated by the musician. It is also possible to use information from every MO available at that moment within the whole model.

An algorithm can be formed by one or more single operators. Each operator



can affect either the parameters of the notes or their order. Each operator is applied to the outgoing MO from the operator before. Set in its header, the algorithm is applied to the range of notes as each note occurs. The header also defines the kind of parameter to which the operator expression must be applied — pitch, duration, intensity (key velocity), MIDI channel — and the order of the affected notes. Let us examine two examples of such algorithms.

(1) *Specular inversion*. This can be done by means of a reference note, called the mirror note; that is, every processed value of pitch is symmetrical to the mirror note. The following is an operator to invert the pitches with respect to C-sharp 3:

P: 1, \$, [Theme, 1], 2 * C#3 - ?

where P stands for pitch; 1, \$ stands for "from the first to the last note of the input MO"; [Theme, 1] determines the use of information from the Theme MO (not from the MOs joined to the input places, which are the defaults and need no explicit specification) starting from the first note (1); and 2 * C#3 - ? is the expression that calculates the new pitch values. The ? is a metacharacter that represents the value of the parameter (in this case, the pitch), assigned to the current processed note.

Other algorithm headers affecting note parameters are:

C (channel) = operator on MIDI channels,
D (duration) = operator on duration, and
L (loudness) = operator on intensity (key velocity).

(2) *Inversion of the order (retrogradation)*. The inversion operator rewrites the input MOs backwards: I: 1, \$, where I stands for invert, and 1, \$ stands for "from the first to the last note."

Other algorithm headers affecting the order of notes are:

K (kill) = operator for deleting notes,
S (save) = operator for preserving notes,
M (multiply) = operator for multiplying notes, and

R (rotate) = operator for rotating notes.

Figure 3b shows a very simple net that can transform MOs. Here, we can join any algorithm we wish to the transition Alg, decide if we want to enable the execution of the Theme and/or Theme.mod MOs (setting the specific place attributes), and decide if the MO obtained from the transformation (Theme.mod) should be volatile or stored in a file. In this case, a specular inversion algorithm may be P: 1, \$, 2 * C-sharp 3 - ?

Firing and timing. PNs are particularly suitable for describing concurrent processes and controlling their synchronization (or lack thereof). The behavior of the net implicitly determines timings. We can synchronize MOs simply by suitably structuring the node connections. The structure of the net determines the firing sequence.

The firing sequence changes with different executions of the net, so there is no correspondence between a net and its firing sequence. We can have many transitions qualified for firing at the same time and not know which one fires first. The net does not describe this kind of information. Every execution of a PN model may give a different firing sequence.

Our approach represents timed MOs by places and their transformations by transitions. The firing of a transition has a null duration. In this way, when a token is put into a place with an associated MO, the token cannot be considered for the firing of transitions connected to the place until the associated MO has ended.

Hierarchy and macros. In this section, we introduce two new concepts: *hierarchy* (refinement) and *macro*.

A refinement, called a *subnet*, is a PN that gives a more detailed description of a node (either a place or a transition) of the upper level of abstraction. If we choose to define a subnet associated with a place P, then the daughter net must have two special places: an input place In and an output place Out. Input arcs of P become input arcs of In, and

output arcs of P become output arcs of Out. Transitions can be refined in the same way. We can also model recursive nets (that is, nets that contain themselves as refinement nodes). To avoid endless recursive expansions, we must assign a recursion level number to every recursive net.

Furthermore, when we design a net model, we often have to create many similar nets that differ in their place attributes and algorithms but are identical in structure. With a macro net, we can use just one net as the base model and then modify its attributes and/or algorithms as needed. This enables the musician to create personal libraries containing commonly used nets. We can do this by writing a modifier list for every subnet place or subnet transition we want for the net. This modifier list is loaded into memory and actualized before execution of the model.

Figures 4a, 4b, and 5a represent a deterministic model based on the "Are You Sleeping" theme, while Figures 4a, 4b, and 5b represent a nondeterministic version of the same melody. (The alternative depends on Figures 5a and 5b. Figures 4a and 4b are deterministic if they are considered stand-alone nets.) The IEEE net is the most abstract net. The Are You Sleeping place is then refined by the homonymous net with Input as input place and Voices as output place. The input arc to the Counter place has weight 3; this means that the firing of the transition will put three tokens in the Counter place. The Rest 4/4 place has a 4/4 rest as an associated MO.

To expand the Voices place with a macro net, we can use one of the two nets in Figure 5. In the first case, we obtain a deterministic execution of the model; in the second case, we obtain a nondeterministic one. Stated another way, we cannot fix the firing sequence, but the firing sequence is determined by Scoresynth with a pseudorandom uniform distribution of the order.

The Input place has no token as initial marking because it will receive tokens from the upper level of the model (in both the Are You Sleeping and Voices nets). Figure 6 shows the parameter list for invoking the Voices macro PN

PROJECT OVERVIEWS



(note that we invoke only the actualization of algorithms associated with the four transitions of the macro net here, but we can do the same with respect to all the possible attributes of the PN).

Figure 1 shows the map of the MOs obtained from the IEEE model execution. It contains four different instances of the Theme.mod MO belonging to the Voices net put on different MIDI channels. These MOs differ from one another because they are generated by different algorithms. The other MOs of the model are not present in the map (or the output score) because Theme was set as an MO that is not meant to be played and Rest 4/4 produces a delay, since it is a rest MO and not a MIDI event.

Transforming music structures. Our research has shown that music structures can be transformed simply by

- modifying markings, MIDI channel specifications, and capacities of places;
- modifying labels of arcs;
- modifying MOs associated with places;
- modifying algorithms associated with transitions;
- modifying the structure of PNs; and

- executing nondeterministic PN models many times.

While editing PN structures and parameters affects causal/structural relationships within the architecture of scores, editing both MOs and algorithms affects the basic information units and their transformations. We believe that the Scoresynth system provides a powerful means for describing and processing music in a way that is closer to music thinking and perception than common music notation is. ■

Acknowledgments

We thank Enrico Bianchi, Antonio Camurri, Giovanni Degli Antoni, Giampiero Jacomini, Antonio Rodriguez, and Renato Zaccaria for their contributions to previous research projects and experience related to Petri nets and music. We particularly thank Sergio Scolaro for his relevant contribution to the design and development of some features of Scoresynth Rel. 2.0 (especially the graphic interface and PN macros). Moreover, we especially thank members of the staff of the Laboratorio di Informatica Musicale at the University of Milan for the scientific and technical support they provided during our research.

This research is partially supported by the Italian National Research Council through the Music Topic LRC C4. Grants No.

89.00031.69 and 90.00678.PF69: "Intelligent Music Workstation," Goal C: Sistemi Avanzati di Produttività Individuale. Subproject 7: Sistemi di Supporto al Lavoro Intellettuale, Finalized Project Sistemi Informatici e Calcolo Parallelo."

References

1. J.L. Peterson, *Petri Net Theory and The Modeling of Systems*, Prentice Hall, Englewood Cliffs, N.J., 1981.
2. G. Haus and A. Rodriguez, "Music Description and Processing by Petri Nets," *1988 Advances Petri Nets*, Lecture Notes in Computer Science N.340, Springer-Verlag, Berlin, 1989, pp. 175-199.

Goffredo Haus, scientific director of the Laboratorio di Informatica Musicale at the University of Milan, does research in the university's Department of Computer Science and composes music by computer methods at both the score and sound level. He helped found the Italian Association of Musical Informatics.

Alberto Sametti is the software manager of the Intelligent Music Workstation project at Laboratorio di Informatica Musicale at the University of Milan. He works on Petri net applications to music and is a scientific consultant to Apple Computer on software development and validation.

