HAL
open science

An Object Oriented Visual Environment for Musical Composition

Gérard Assayag, Carlos Agon, Joshua Fineberg, Peter Hanappe

To cite this version:

Gérard Assayag, Carlos Agon, Joshua Fineberg, Peter Hanappe. An Object Oriented Visual Environment for Musical Composition. ICMC: International Computer Music Conference, Sep 1997, Thessaloniki Hellas, Greece. pp.364-367. hal-01161427

HAL Id: hal-01161427

https://hal.science/hal-01161427

Submitted on 8 Jun 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Serveur © IRCAM - CENTRE GEORGES-POMPIDOU 1996, 1997. Tous droits réservés pour tous pays.*All rights reserved.*

An Object Oriented Visual Environment for Musical Composition

Gérard Assayag, Carlos Agon, Joshua Fineberg et Peter Hanappe

ICMC: International Computer Music Conference, Thessaloniki, Hellas, Greece, Septembre 1997
Copyright © Ircam - Centre Georges-Pompidou 1997

Abstract

OpenMusic is a visual programming environment for composers based on Macintosh Common Lisp. It extends the functionalities of PatchWork to full object oriented programming and provides a high level control on the musical form.

The Musical Research Group at Ircam has conceived and developed computer based systems to help composers treat certain aspects of musical composition. While there are many different aspects to composition, we have restrained ourselves to those which are directly linked to the symbolic manipulation of musical material.
Thus, problems such as performance and real time processing do not form an integral part of our work. The basic substance we treat are models, compositional techniques at their more basic level : treated as a universe of forms, structures and relations.

Our group has specialized in visual programming languages for musicians. We have tried to make the transition between various levels of compositional conception and realization as continuous as possible. Our current work has taken the form of a new environment provisionly named OpenMusic. OpenMusic is the continuation of PatchWork [Laurson 96] , an environment by Mikael Laurson and Ircam which already made use of visual programming.

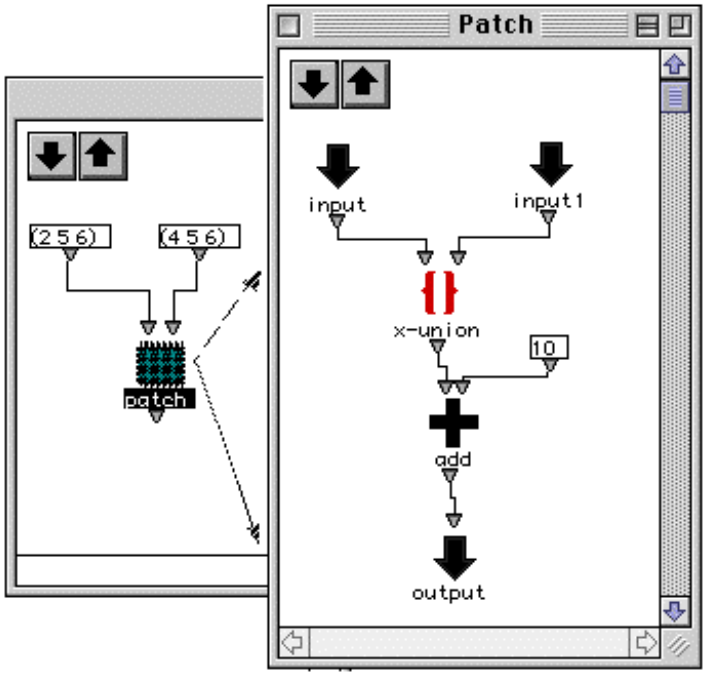
PatchWork is considered one of the most powerful and versatile program currently available to aid a composer in the creation of his music. This program has become indispensable in the daily work of numerous composers from the entire gamut of aesthetic tendencies. However, as composers have gained experience with the program and as new types of applications have become feasible, the limitations, both technical and theoretical, of the current form of PatchWork have necessitated the envisioning of a major reworking of the program.

Currently, PatchWork is made up of hundreds of incremental improvements built directly upon previous versions, leaving tremendous disorder and making the program difficult to maintain. In this context, major structural changes are extremely difficult and may produce innumerable hidden consequences.

The current version of PatchWork is centered around the patch editor window ; in this window functional modules may be connected in any way the user desires so as to perform calculations of almost any type. These modules are made available to the composer either as standard PatchWork kernel boxes, as standard Common Lisp functions, as parts of various user-libraries, or may be programmed directly by the user in Common Lisp. Unfortunately, this great flexibility is confined within that one patch. While not impossible, it is extremely difficult to collect information from one patch for exploitation in another. Temporal aspects, too, may be controlled within a given patch but are difficult to transmit from one patch-window to another. Even the formats of objects make inter-communication difficult (for example: a chord may be defined by a list of midicents; a list of lists containing midicents, velocities, durations, etc.; a list

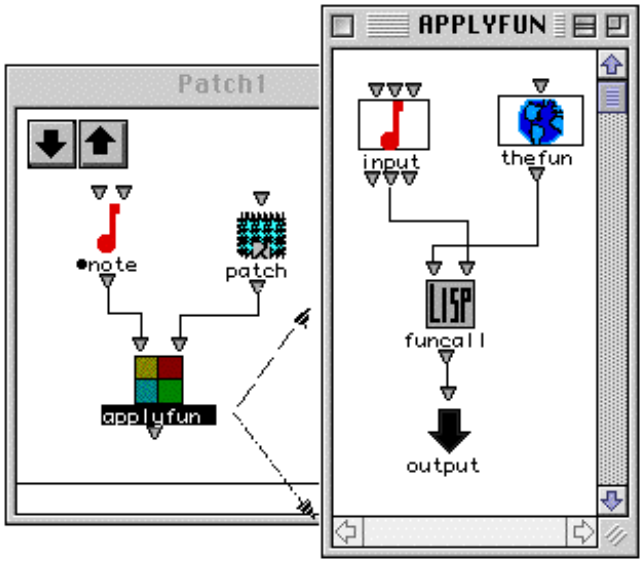
of note objects; a chord object; or a chord-line object). These different formats permit an individual chord to be calculated with as much or as little complexity as required, but create endless conversion problems when two different forms of this same type of musical object must be placed into relation.

For a new environment to supersede these problems it would have to have the capacity to place various patches into more supple relationships, to dynamically create object definitions, to facilitate the communication between various forms of the same musical structure and to be able to place all of this into an easily editable temporal context.

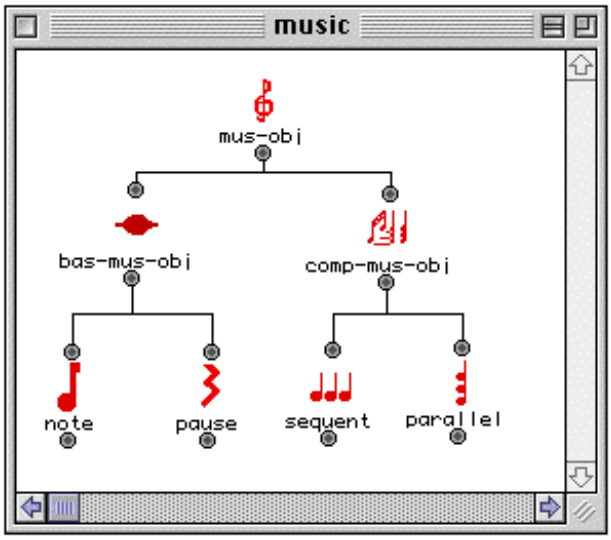


A patch can be used as an abstraction in an other patch.

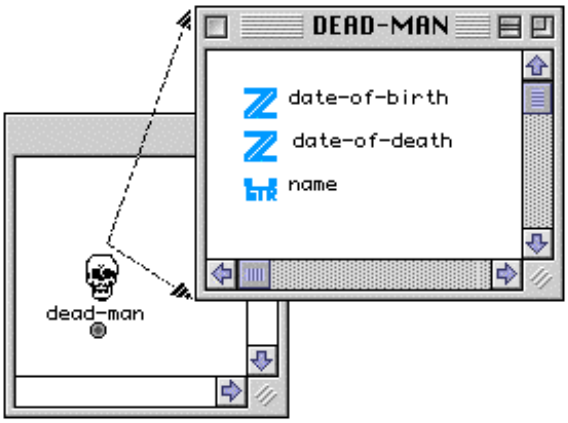
The basic patch editor which formed the center of PatchWork has been completely rewritten for use in Open Music. This coherent and efficient patch editor has allowed the integration of important improvements in programming power: including object oriented tools such as the easy graphical definition of object, classes and methods, based on the underlying CLOS model [Steele 1990].



A patch may be passed as a functional argument to another patch.

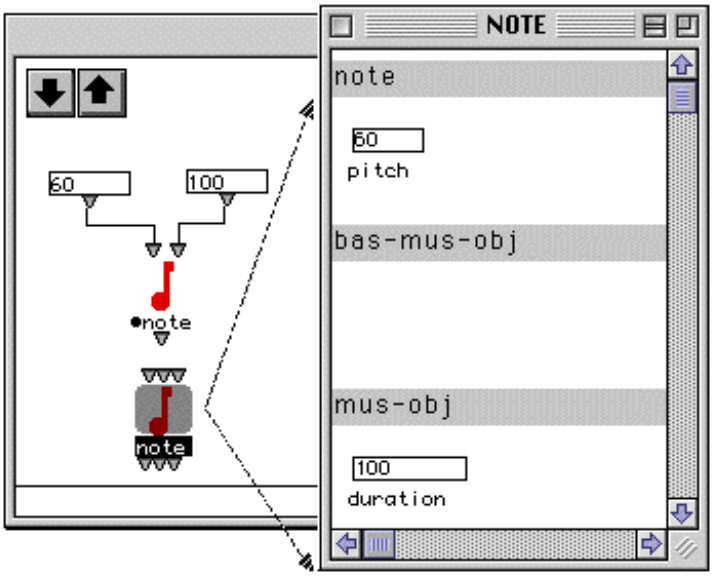


A class hierarchy .

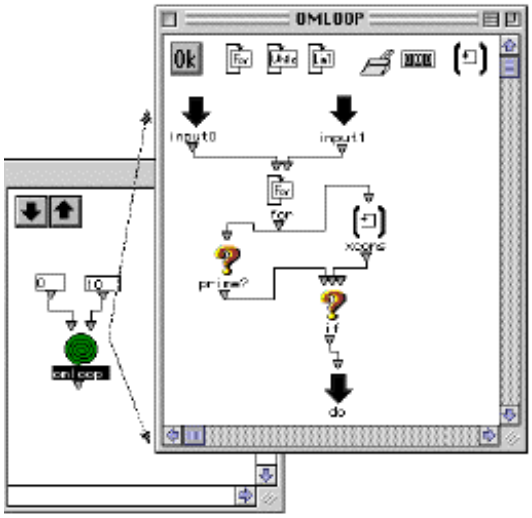


Access to the slots of a class.

Other Macintosh standard features such as scroll bars, balloon help and drag-and-drop were finally implemented. Additionally, it is now possible to create sophisticated loops and tests which are nearly impossible to perform in the current PatchWork without programming Lisp code directly. Many other new features have been added including an inspector window that allows various aspects of modules or patches to be defined.

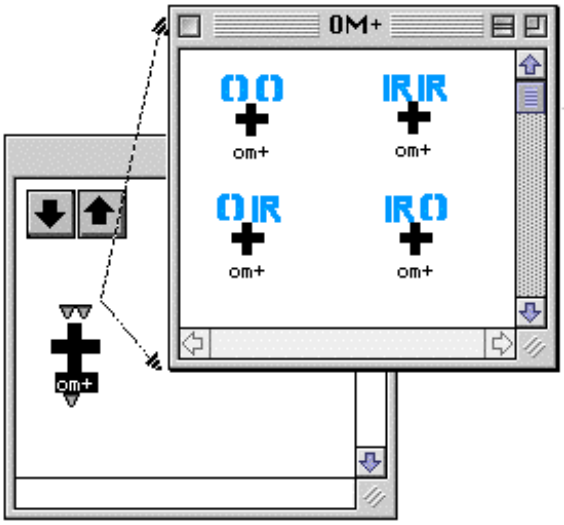


From the class " note " an instance is generated and materialized as an icon.



A loop computing the list of prime numbers between 0 and 10.

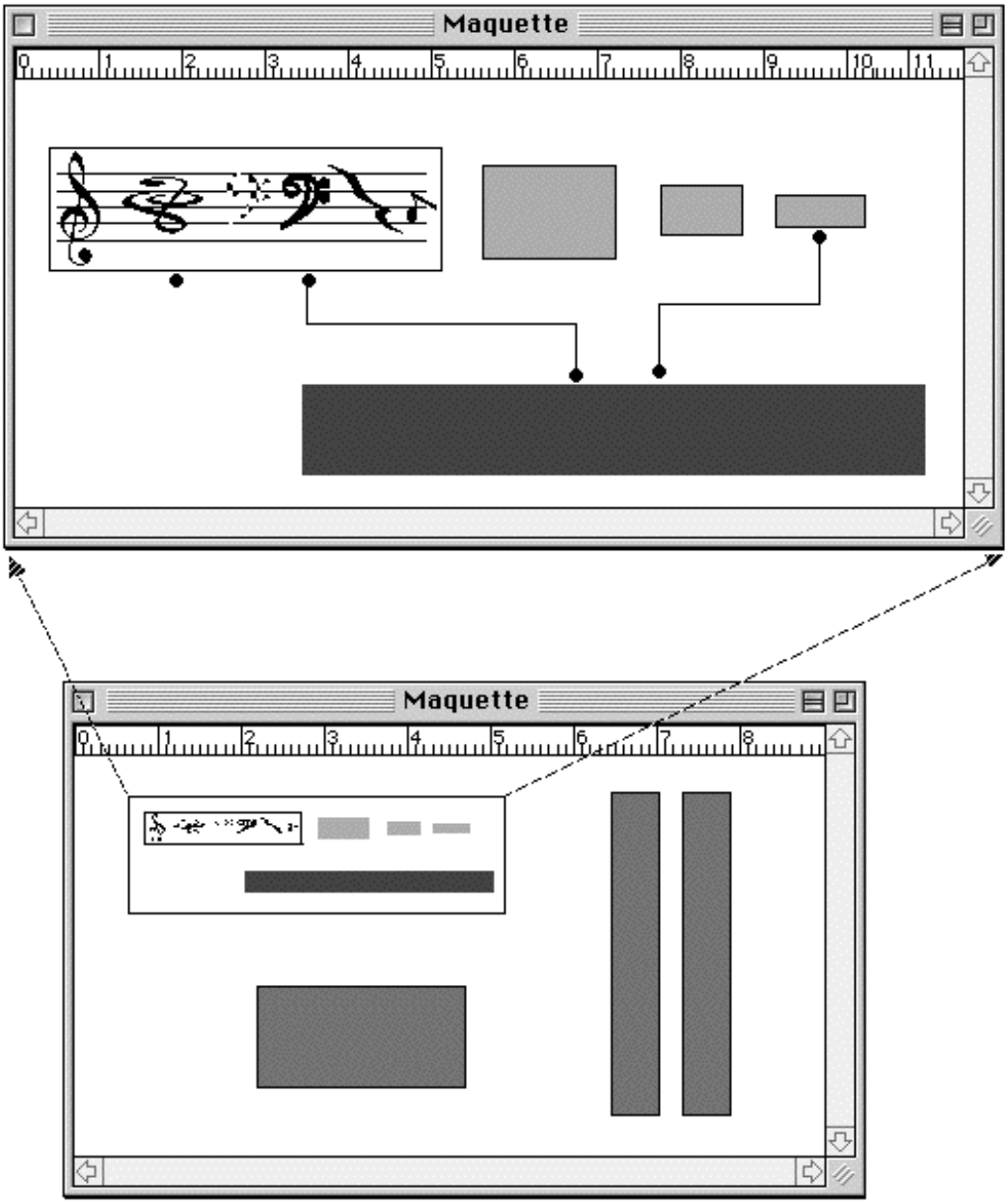
These changes in the infrastructure will allow a tremendous degree of flexibility in the visual customization of functions without writing code. It will be possible to cut and paste entire musical objects directly into the default values for a function, allowing composers to rapidly modify defaults for a given piece or even a section of a piece. Although the current patch editor is very effective for treating lists, once a certain complexity of material is reached the use of objects becomes essential. OpenMusic provides a visual interface for creating new classes by derivation of existing ones, instances of these classes, and generic functions acting upon these instances and switching between the appropriate methods depending on the type of the input. Thus the basic battery of functions will be much more musically direct than in the past. For example a transposition module would be able to look inside any musical object (note, chord, chord sequence, etc.), find the appropriate slot, perform an addition or subtraction and recreate a transformed copy of the original object with its notes transposed. This would be in a uniform manner using the same generic transposition module ; if the user required a particular non standard treatment he could simply graphically create a new user defined method of transposition that would apply to the case at hand.



*A generic addition function and its four methods defined for several combinations of the classes number and list (R and **()**)*

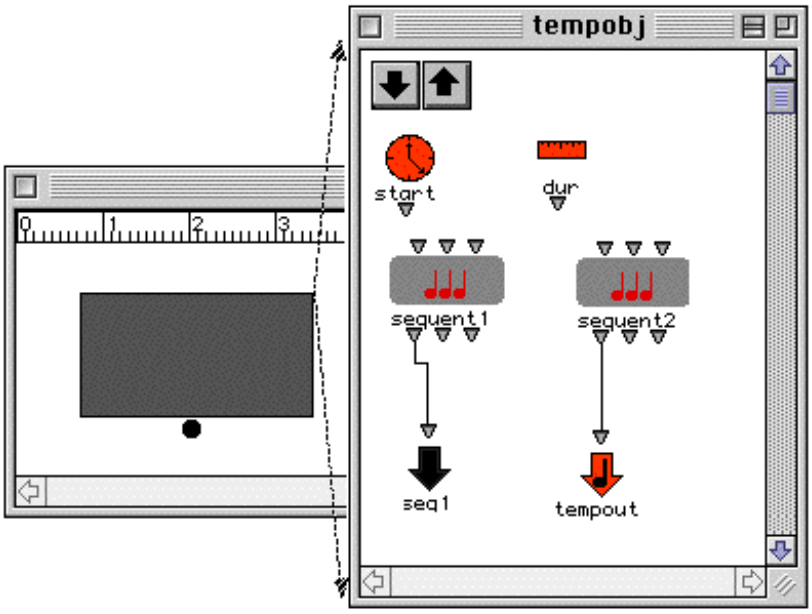
The second major problem in PatchWork was its limitations in establishing links between patches

and allowing the easy juxtaposition of musical materials in time to simulate complex musical sequences. The solution that we have proposed in OpenMusic is the Maquette Editor. This specialized interface allows the creation of blocks placed in spatial and/or temporal relationships. These blocks are linked to patches. They may contain inputs or ouputs that are defined from within the patch associated with each block. Besides these inputs and outputs which allow interconnections to be made between different patches, directly within the maquette, each patch editor window contains two special inputs and one special output. These inputs and output are used to communicate temporal information to the patch or to recover it for the maquette editor. The inputs, which are automatically present in each patch, though need not be used, communicates values corresponding to the position and duration of the patch's block in the maquette editor. This value may then be used in the calculation of the patches result [Lindemann 90]. The special output is used to collect the musical structure provided to the maquette by the patch. These objects may be of any type from a series of real durations to complex polyphonic sequences in shifting tempos and meters; however, all types of musical or even just temporal objects must ultimately be connected to this output for their results to be available from the maquette editor window. The temporal durations of these objects, upon evaluation from within the maquette, will cause the block to be rescaled horizontally for display. This 'real-duration' display reflects the actual duration of the musical objects created from the patch contained in the block (in relation to the display scale, of course). This display allows the user to obtain a clear graphic representation of the actual durations of various musical materials and easily compare and juxtapose them. It is also possible, once a real-duration has been calculated, to scale the entire block to a new horizontal-size; thus rescaling proportionally all the durations contained therein. If the musical object was notated symbolically the user has the option of rescaling through either a change of tempo or a re-quantification. This 'scaled-duration' display is always shown as a superposition over the real-duration display. Thus allowing the user to keep in mind the degree of scaling that has been performed.

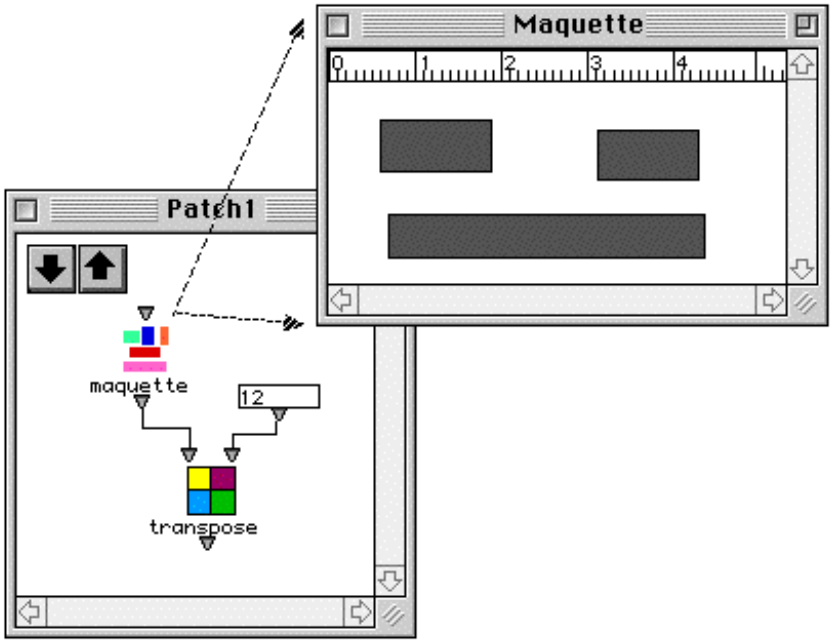


The maquette below contains another maquette (above) and three patches.

Other scaling tools will also be available including time-placement arrows, which allow, for example, the beginning of one block to be linked to the end of another. This type of link will mean that any subsequent change in the length of the latter block will shift the start position of the former. More complex links, requiring rescaling are also possible, for example, a sequence could have its beginning and end linked to another sequence; thus a change in the second sequence's duration will automatically force a rescaling of the first. Since the scaled duration is always displayed superposed on the real-duration it is easy to keep track of the effects provoked by these links. To avoid confusion in the hierarchy, these links are displayed as arrows with a point at the end representing the master and the arrow at the end representing the slave.

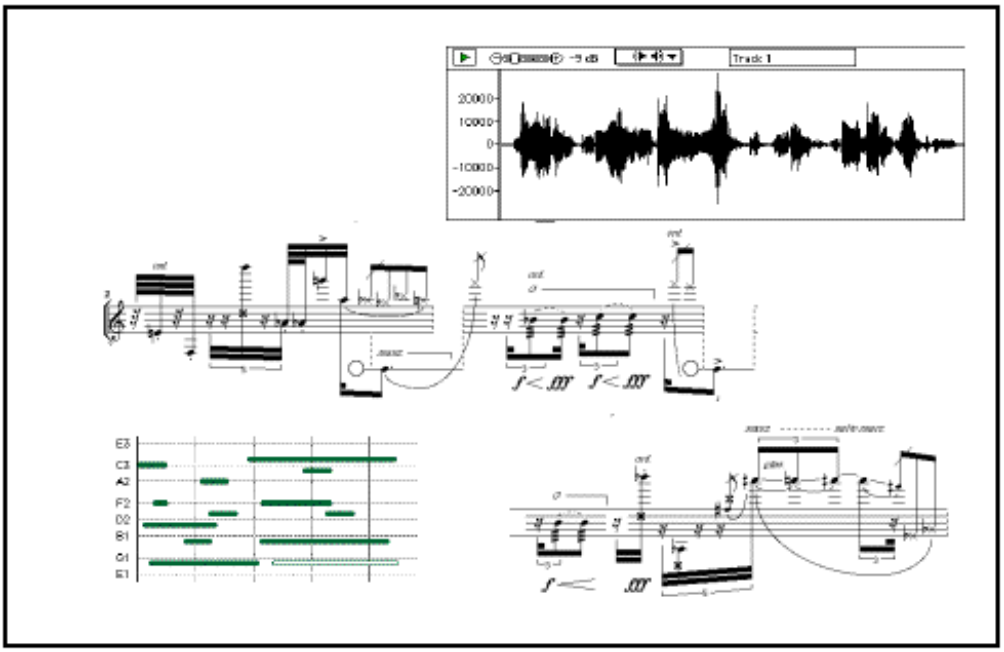


When a patch is dropped into a maquette, it receives two special inputs that represent its start time and its duration. The musical structure computed by the patch and relevant to the maquette is connected to the special output " tempout ". The other output " seq1 " is an additional output that can be used to communicate with another patch in the maquette.



A maquette has been dropped into a patch. Its musical content can be provided to other operators in order to be further transformed.

It will be possible to output either an entire maquette or only the selected blocks to a music notation editor. If only selected blocks are used the spaces between the blocks will be respected, but the first block will be shifted to time zero, full maquettes will strictly respect the placement of each block.



A maquette is a container where heterogenous musical data may be arranged (score excerpts from " Huskless " by Paul Steenhuisen)

Conclusions

PatchWork had become an essential tool for all composers using complex materials. With the computer to calculate material of an arbitrary complexity composers could be free to experiment and refine even very complex materials into a satisfactory form. In spite of this great advance, PatchWork's limitations, both conceptual and technical have necessitated the creation of a new environment to allow composers to work more easily with multi-dimensional material and to experiment with the interactions and formal relationships between all sorts of musical materials. Currently, most composers perform the later, formal stages of composition at the table: current tools often being more hindrance than help. With Open Music, this stage of the compositional process will become susceptible to computer assistance. Not only will the current functions of PatchWork become more powerful and accessible, but the maquette editor should also allow the same type of experimentation and refinement that PatchWork offered for the generation of material at the formal level. The new musical score editor should go even further, allowing the composer to begin to transform a multi-layered model into a more refined and perfected form; thus advancing significantly closer towards a finished score. While there are certainly limits to what aspects of the compositional process may be practically or even ideally helped through the use of a computer, Open Music should be able to go a long way toward removing the keenly felt limitations of PatchWork. This will be done while still maintaining the open ended and flexible nature that have made PatchWork successful and such an indispensable tool to so many different types of composers.

References

[Assayag 95] Assayag G., "Visual Programming in Music." Proceedings of the ICMC 96, Banff, 1996.

[Assayag & al 93] Assayag G., Rueda C. "The Music Representation Project at IRCAM." Proceedings of the ICMC 93, Tokyo, 1993.

[Laurson 96] Laurson, M. " Patchwork, A Visual Programming Language and some Musical Applications " Sibelius Academy, Studia Musica Ndeg. 6, Helsinki 1996.

[Laurson & al 90] Laurson, M., Duthen, J. "A compositional environment based on Preform II, PatchWork and Esquisse." Proceedings of the ICMC 1990. Glasgow 1990.

[Lindemann 90] Lindemann, E. "Animal: A Rapid Prototyping Environment For Computer Music Systems". Proceedings of the ICMC 1990. Glasgow. 1990.

[Steele 1990] Steele, G., " Common Lisp The Language ", 2nd edition, Digital Press, 1990.