E-SCAPE
An Extendable Sonic Composition and Performance Environment

Tim Anderson
Music Technology Group
University of York
Heslington
York Y01 5DD
Email: tma@phc.c.york.ac.uk

ABSTRACT: This is a customizable, hierarchical graphical system for the
collection of sound and transformable data of musical and sonic objects. Sound can be
produced by interfacing to any soundscape DSP or commercial synthesizer hardware
via a device database containing translation algorithms. Abstracted data patterns can
effect control of sound objects at any level, on any scale, with device-independent
interfaces, and produce control parameters available from the database. Algorithmic
processing objects can be constructed by the user from networks of primitives,
presented graphically or textually. They can also describe device configurations which
are downloadable to hardware.

This flexible "loosely coupled" [Loy 86] software system enables composers to create, structure,
process and display sonic and musical data structures of any size and complexity in ways not
necessarily foreseen by the system designer, and to realise sound on a variety of external hardware.
The system and its user interface can be customised and extended. Development is on
Smalltalk-80, ensuring portability as well as other advantages of object-oriented languages [Pope
89, Pfeifer 84]. All sound objects can be viewed and edited graphically at any level of detail.
Events are displayed as named slots, or shown in the system in detail. Musical and sonic parameters
are abstracted from the sound device's output interface in use, and any device can be added into the
system by defining an appropriate protocol for its control. Music data can also be output as textual
scores and orchestrations for performance by a real-time MusicN system. This a combination of
one specifying micrographic timbral details, can be performed on any equipment whose functionality
is a superset of the "original" hardware, e.g. if a piece uses a filter sweep, then a device must at least
implement a parametric filter. This is not easy, but is an important goal if composers are to establish
an electroacoustic score notation capable of analysis and reinterpretation as is conventional
music notation.

System Structure: The system follows the object-oriented paradigm at all levels: collections of
sound events, data streams controlling musical parameters, are self-contained functional objects,
which can be built into hierarchical and recursive structures of any size and complexity. A
complex-event-object contains a scheduling-object which calls any number of control-
object, processing-objects and further child event-objects. Instead, each primitive-event-object
directly produces sounds. Each knows its state-time relative to its
parent, and can report upwards with other data needed by its parent such as the number of notes
producing in each note window. A primitive-event-object has a setup-object which references a
sound-producing instrument-object on a particular external hardware device. The instrument-
object not only specifies a sound producing "patch" configuration on the device, but also derives from
this the primitive sonic parameter (PSP) which are available for control by the composer on
this instrument on this device. Some PSPs may be fixed, others are t-value parameters such as
"attack-time", which may not be available on a device during a primitive event.

Each different configuration will have its own set of PSPs, all with default values for patch and
volume will almost always be available, but others such as "FM-index" will be more instrument
specific — e.g. filter-on resonance — will not be available on a pure FM instrument.

An important factor for the user is that only sonic parameters are overtly available or displayed, e
messages peculiar to the interface are not shown. For example, an instrument on a MIDI-controlled
device will not display "aftertouch" or "velocity" parameters, as the sonic effect of these
messages depends on the device and patch configuration in use.

ICMC GLASGOW 1990 PROCEEDINGS
212
Rules in a device database decide how to communicate PSPs, often using these messages in conjunction with its knowledge of the device's patch configuration. For instance, velocity can be used to transmit fixedPSPs such as velocity attack-release aftertouch to control the patch such as filter frequency, vibrato depth, or FM-index. This shields the user from musically irrelevant details and enables the same PSPIs to be communicated to many different instruments and/or devices.

Patching-event-object also contain any number of actions which they initiate at onset and offset. Common onset actions are to "start-a-note" and to set the value of any fixedPSPs. The low-level protocol invoked by this message depends on the particular hardware device and interface used to implement the instrument-event-object. The patch-event-object is implemented on a MIDI-controlled synthesizer device, "start-a-note" would send a MIDI note-on message at the onset of the patch-event-object. The same patch-event-object could use instruments on another eg custom DSP device - only the low-level implementation of "start-a-note", "pitch", and "FM-index" would change.

All events may also contain even-control-object if which globally affects one of the PSPIs of the instrument-event-object used by the event. An even-control-object consists of an abstract I/O data pattern list of ordered vs value pairs and two parameters: "start-time" assigned to the f data, and an available DSP assigned to the f data. Each DSP value in order is a low-level message is sent at its associated start-time. Event-control-objects are a subclass of the more general control-objects. These contain an n-dimensional data pattern list being model common, with each axis assignable to: time, a DSP, or a complex-tonic-parameter (CSP) which control the activities of processing-objects (see below). Data patterns exist in a library, and are created and stored abstracted from any parameters. Thus, the same pattern generator can be used to control any multiple or any single parameter at any level from microscopic control of a temporal or spatial detail to each patch, to global control of PSPIs such as tempo, pitch, volume, or CSPs such as "flux-

Processing-objects take in any number of data patterns from the library, or directly abstracted from the output of control-objects or event-objects and can output any number of new or modified data patterns to (all) new control-objects which can control many simultaneous parameters of a sonic or musical structure. For example, timing data (event start time from the scheduling-object of a specific level) complex-event-object could be input to a processing-object, and 2 processed data patterns output which are used by control-objects to control pitch reflections of each low-level patch-event-object.

The user can construct arbitrarily complex processing-objects hierarchically from primitive processing and rule units for use "patch-transport" complex's main. Takes from a library, and store them as new single units in the library. Compares can thus build retrieve, or swap (with other composers) complex processing-objects for any kind of algorithmic composition or sound construction. Manipulation and construction is via variable graphic editors (Bryte 86, L systems 89, Puckette 89, Model 89). Smalltalk code, or external modules f is used in C. Data inputs to processing-objects can be named (e.g. requires "on-time" data from another object), or can be specialized controller-objects with CSPs. The data patterns existing at any point within the processing-object can also be named. A CSP and displayed graphically. If a processing-object contains "audio output" units, it can describe a hardware patch configuration. It is defined as a sound-generating structure and its control inputs. This configuration definition can be sent to the device in use, or transmitted into a text orchestra for Musec. Thus there is a uniform user interface for instrument design, control processing, and event location editing.

A complex-event-object's sub-event-objects is a list of the instrument-event-objects of a particular complex-event-object. This includes all the sub-event-objects of a particular complex-event-object. An event-context-object's complex-event-object can only be assigned a DSP which is common to all the sub-event-objects' instrument-objects. Pitch and volume will almost invariably be common; such things as vibrato depths and speed less so. However, all the sub-event-objects' of a particular complex-event-object, eg, using L systems (implemented in a MIDI-controlled FM synthesizer) or configurable DSP system, then a large number of FM-specific timbral control PSPIs would become available for global and display of even-context-objects at this level.

ICMC GLASGOW 1990 PROCEEDINGS 210
If more detailed terminological content is desired, an event-control-obj can be used to lower down the event hierarchy - ultimately, all the PSPs of an instrument-obj are available for control and display (at the lowest level) by a primitive-event-obj.

**Device Database.** Instrument-objs reference a central device database which is queried by the user to match the hardware system in use. Each hardware device in the system is controlled by a device-object which contains:

1. A library of device patch configurations for use by instrument-objs which are implemented on this device.
2. A list of PSPs and fixed-PSPs available on each patch. More complex devices may implement the use of rules which determine the available PSPs from the patch configuration.
3. A series of parameter buffers which may be a large number of PSP values affecting possibly many concurrent events in a series of low-level messages appropriate to the device in use, and take into account the overall system configuration.

a. Rules which may merely be mappings, or sophisticated algorithms which utilize processing-obj to convert PSPs and their ranges of values to one or more channel patch parameter messages, probably with different values. For example, a device-obj using a hardware device which responds directly to the parameter may 'convert' the PSP 'vibrato-rate' to the patch parameter 'vibrato-rate'. A stream of PSP messages from a device-control-obj (log with values from 10% to 50% of the pitch) of the event-object whose control it would result in a similar stream, but with appropriate absolute values for the device. The device-obj also reports back to the calling object if the amount of vibrato demanded cannot be met by its device. On the other hand, a device-obj may control a device which has no vibrato message, or, the composer may demand a vibrato shape, speed, or depth, which the device cannot itself perform. It may still be able to implement the device by converting its values into a time-varying series of patch parameter messages controlling pitch offset directly with appropriate values. Thus each single PSP vibrato value would result in a stream of pitch change messages.

b. Rules to convert channel patch parameters into specific low-level messages on the appropriate port (e.g. MIDI, RS232, DMA, SC35) and protocol (e.g. MIDI, or custom). If the 'device' is a Master timing time system, the low level message may be static text written to disk for subsequent transfer. Again, some programming may be necessary to cope with the limitations of some interfaces.

c. Rules to alter low-level messages to suit the device system configuration in use. For example a DSP or MIDI-controlled device may be configured as several logically independent devices with individual control channels, but with some overall limits on the device as a whole, e.g. regarding the total number of simultaneous notes, or the bandwidth of continuous controller information. Algorithms can again help to overcome any limitations of an interface in use - MIDI is the obvious example. A composite may demand a series of overlapping unpitched events, which change pitch, and which all use the same instrument implemented on a MIDI-driving device. Each note may not then be sent to a different logical (or actual) device with independent pitch bend and/or sysex, frequency control. This requirement must be determined before playback to allow setting up copies of the patch configuration on the necessary number of devices. If a composite does more impossible demands, options can be available to ben cope with the limitations, e.g priority note allocation [Anderson 87].

Some devices may be able to have certain performance data as well as the normal configuration setup data loaded and stored prior to performance to ease bandwidth limitations problems. It is to be hoped that this capability will become more prevalent as increasing complexity and 'control intensity' [Miller 83] demands create system bandwidth-limitation problems. Essentially the central system may be able to re-distribute much of the complex performance data to a number of devices, then merely "conduct" their playing (cf. MTC cue lists).

The graphics presented with this paper show a variety of example views, and structures. This paper has been abbreviated in order to compile with ICMM space requirements. The full text and illustrations are available from the authors.

ICMC GLASGOW 1990 PROCEEDINGS 214
APPENDIX: Summary of System structure (as seen by the user)

Complex-event-object
  .Setup-object
    Device-objects
      Instrument-objects ➔ Device
  ➔ Database
Scheduling-object
  Complex-event-objects
  Primitive-event-objects
  Processing-objects
  Control-objects ➔ Processing Library
  Event-control-objects ➔ Pattern library

References

Blythe, D., J.Kirwan, D.Galloway, and M.Sinelgrove. 1986. "Virtual Patch-Coins for the
Kawai synthesizer." Proc. ICIC’86.
ICIC’86.
ICIC’86.
Timbre Production." Proc. ICIC’87.
Music System." CML 13(4): 54-64.
ICIC’86.
Object-Oriented System." CML 13(2): 71-76.
ICIC’87.
Pilansky, L., D. Rosenboom, and P. Birk. 1987. "VWSDL: Overview (v. 1.1) and Notes on Intelligent
Tools and Knowledge Representation for Music." Proc. ICIC’86.
31-36.

ICMC GLASGOW 1990 PROCEEDINGS